ARIS

DISTRIBUTED INSTALLATIONS COOKBOOK

VERSION 10.0 - SERVICE RELEASE 14

April 2021

Important disclaimer: This document provides you with the knowledge for installing ARIS in distributed environments. For acquiring that knowledge Software AG recommends that you attend the training course ARIS Server Installation available via http://softwareag.com/education, or ask for an ARIS installation service by Global Consulting Services.

TABLE OF CONTENTS

1 Introduction & Overview

1.1 About using a distributed ARIS installation

The goal of this document is to provide information about how to manually set up a distributed (or "multi-node") ARIS installation. In a distributed installation, the ARIS components (or "runnables") are located on several machines instead of on a single machine. You can use a distributed installation for the following reasons:

Available hardware resources & cost: In some more demanding usage scenarios, the total hardware requirements (in particular the number and performance of CPU cores and available RAM) of all ARIS components might be more than what a single machine (or at least a single **affordable** machine) can handle. By installing the components across several nodes, you can use several smaller (and thus cheaper) machines instead of one larger and perhaps prohibitively expensive machine for an ARIS installation.

Requirement for high availability: When installing ARIS in the standard way on a single machine, the availability of your ARIS system will depend on the availability of this single machine. ARIS will become unavailable for its users if that machine becomes unavailable (for example, due to a hardware defect, power or network outage of the machine, power or network outage of the entire data center, etc.). By installing the ARIS components on multiple machines (which might even be located in different data centers to protect against the failure of an entire data center) , you can keep ARIS up and running in such circumstances.

In addition, if you do not have such requirements and do not plan to distribute the runnables across multiple nodes, this document will give you the necessary information to manually (that is., without the use of the setup) install a **single** node.- A single-node installation can be viewed as a special case of a multi-node installation.

Throughout this document, as a running example, we will develop a distributed ARIS Connect installation. We tried to keep this scenario as close as possible to what you would do in an actual customer environment, but in some places compromises have been made to keep the efforts and hardware requirements manageable. In those places, the compromises will be pointed out so that you know how to adjust when installing an actual production system.

1.2   Structure of this document

The document has several chapters. After a brief overview of the ARIS 9/10 architecture in the remainder of this chapter, chapter 2 continues with an overview of the basic concepts of distributed computing and high availability. Chapter 3 explains all the necessary preparation steps to get started with a distributed installation. Chapters 4 to 7 explain the role of essential backend services in every ARIS installation, Zookeeper, Elasticsearch, the relational database, and Cloudsearch, and detail how to make them highly available. Chapter 8 explains how the micro-service-based ARIS application components can be distributed for high availability. Chapter 9 describes how to ensure that client access to the installation does not become a single point of failure. Chapter 10 gives basic guidelines on how to operate a distributed ARIS installation.

1.3   Prerequisites

It is recommended that you have a good understanding of and some familiarity with the ARIS provisioning framework because this is the main toolset that will be used throughout this document.

To try the sample scenario that is developed throughout this document, you need access to three physical or virtual machines that will act as "worker nodes", that is, they will be the machines on which the ARIS Cloud Agent and the ARIS runnables will be installed. For training purposes, the scenario can be kept small, so that you can run it on a host with four CPU cores and 32GB RAM, running three virtual machines with 8GB RAM each. The example scenario can be adapted for production use, but then a significantly higher amount of hardware resources will be needed.

1.4   ARIS Architecture

ARIS versions 9 and 10 are based on a micro-service architecture, illustrated in figure 1 below. In a micro-service architecture, the functionalities of the application are no longer contained inside a single, monolithic application server, but instead are spread across multiple smaller application servers, or **micro-services**. Micro-services are individual operating system processes that are stateless, that is, they do not store any persistent application state (that is, the actual application data, such as ARIS models, users, documents, comments, etc.). The persistent application state is instead stored in **backend services**. A **load balancer (LB)** is used as a single point of entry for **clients** accessing the application. ARIS offers both web-based thin clients that run in a browser and more traditional rich clients that run as dedicated applications on the client machines.

FIGURE 1: ARIS 9/10 ARCHITECTURE
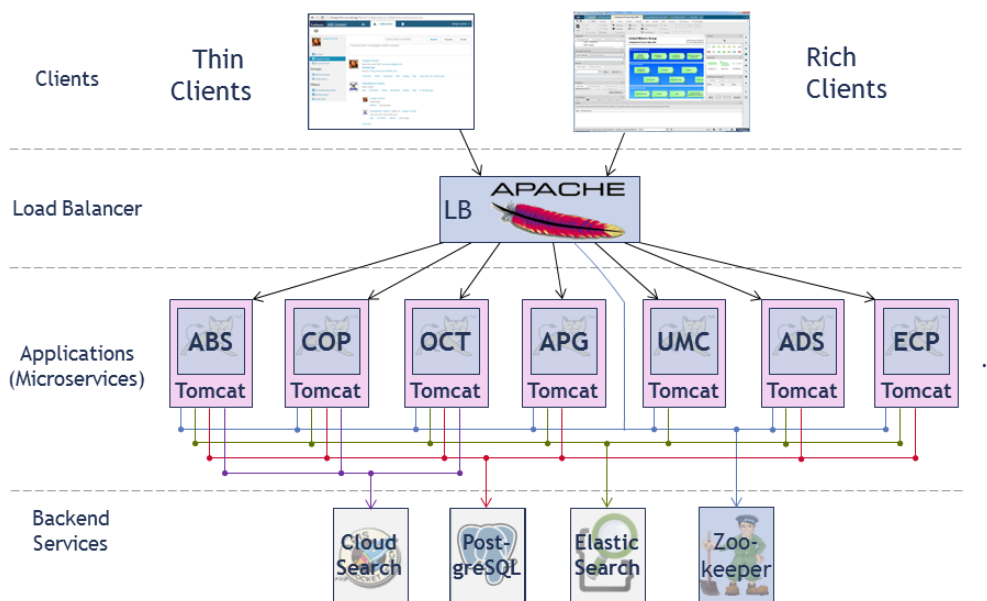
The individual micro-services and backend services can be deployed independently. To make that possible, both micro-services and backend services are packaged as so-called **runnables**, which can be installed, started, stopped, and configured using the ARIS provisioning framework. Depending on the product being installed (ARIS Connect, ARCM etc.) the types of required runnables differ. A minimum installation of a specific product contains a single instance of each runnable, and all runnables are located on the same node.

1.5   ARIS provisioning & setup

The ARIS provisioning framework consists of two main components, illustrated in figure 2 below. ARIS Cloud Agent is a small service that must be installed on all machines that serve as a worker node in an ARIS installation. ARIS Cloud Agent offers a REST API through which it can receive commands from the second component, ARIS Cloud Controller (ACC). The cloud controller is a console application that accepts commands that are run in an interactive shell. It can control several cloud agents simultaneously, which is useful when you work with distributed environments.

FIGURE 2: ARIS PROVISIONING ARCHITECTURE
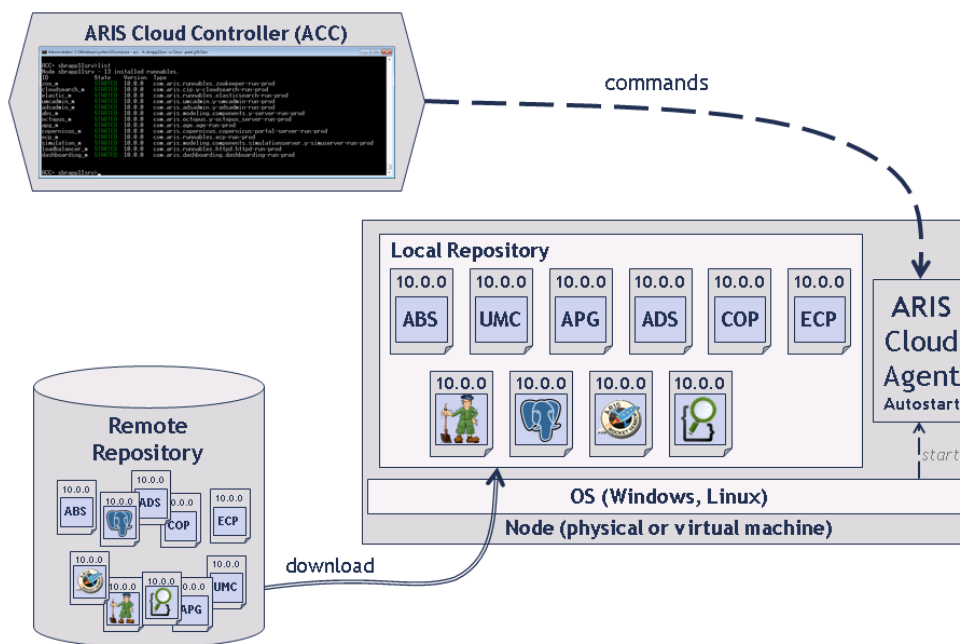
The core functionality of provisioning is to "install" runnables (using the **configure** command), as well as start, stop, reconfigure, or update them. When told to install a runnable, the agent will try to download it from a remote repository, whose URL must be specified beforehand. The ARIS setup also uses ARIS provisioning under the hood. The setup installs the cloud agent as a service on the respective machine. The setup then uses an embedded ACC to execute the commands (in particular the **configure** commands) that install all runnables required for the product. A single instance of each runnable is installed. While the setup is running, it also provides the remote repository, from which the agent can download the runnables and other required artifacts (for example, help files).

For many ARIS usage scenarios, it is sufficient to have a single instance of each type of runnable in the system.

Advantages of distributed micro-service architectures

A distributed micro-service architecture has many advantages over traditional monolithic applications, where all functionality is contained in one big piece of software:

Scalability - If an application handles an increasing load (for example, an increasing number of concurrent users, an increasing amounts of data, etc.), it will sooner or later reach a point where performance degrades,that is response time increases, batch jobs take longer, and the overall user experience suffers. Aside from improving the performance of the software itself, which is a lengthy process and has its limits, the only other option is to make more hardware available to the application. In a traditional monolithic application, your only chance to remedy this situation is to **scale up** your hardware, that is, buy a bigger server. While this will work well up to some point, there are limitations to how big you can make a single server. Besides, even if you are still far away from what is technically possible, once you leave the range of mainstream commodity servers, buying more powerful hardware becomes disproportionally more expensive. This restriction becomes more severe when you are working with software virtualization in your data center or when you are running your software on virtual machines provided by an Infrastructure-as-a-Service (IaaS) provider: Usually there are - strict limits to how big you can make an individual virtual machine. With a distributed application, you can simply add new instances of the individual application components, instead of making the single instance bigger, that is., do a **scale-out**. If the capacity of a single server does not suffice anymore, you can add another server and put some of the components there, instead of buying a new, bigger server.

Higher quality due to improved maintainability - In a monolithic application, there is no technical limitation regarding the interaction of application components -each component can interact directly with any other.. This makes it easy to create new dependencies among individual parts of the application (often without knowing). Unless very strict discipline is observed when making changes, this can over time lead to a code base that becomes harder and harder to update and maintain. Because anything can depend on anything else, the effects of what appears to be a small local change on the rest of the system become harder and harder to estimate. If instead you are working with isolated components with clearly assigned functionalities (that is, micro-services), you invariably have to interact with other components through dedicated, well-designed, public and usually quite stable interfaces, making the interdependencies clear and isolating the components from changes in others. This reduces the amount of coupling among components and makes it easier to change and test the individual components, thus guaranteeing e good software quality even when adding a new functionality.

High availability - When you have a monolithic application on a single server, if anything happens to that server (for example, hardware failure, network outage, or power outage, etc.), your application will become at least temporary unavailable. If an application is mission-critical for an organization, such an outage can cost a lot. While it is possible to make single servers highly available by using corresponding hardware with redundant components, this is again expensive and there are limits to what can be done. With a properly designed distributed application, it is instead possible to stick to commodity hardware (for the most part) and to achieve high availability by configuring the application in

a redundant way. In this case, when an outage of individual components occurs, the application will continue working (albeit perhaps with reduced capacity).

The remainder of this document will discuss how to leverage the advantages of the micro-service-based ARIS architecture to set up distributed and highly available ARIS installations.

2 High availability overview

2.1 Availability of software systems

A software application that is vital for an organization to do its business should ideally be available to its users around the clock. Availability is usually understood as a situation in which the system is able to fulfil its functions without restrictions. A system is highly available if it can fulfil its functions for a bigger period than normal. Availability is usually given as a decimal number between 0 and 1 that is obtained by dividing uptime by total time, which can also be understood as a percentage. In high-availability parlance, it is common to refer to the number of nines behind the decimal point to indicate the rough category of high availability of the system. For example, an availability of 0.999 would be "three nines" or 99.9% availability. While this might seem like good availability at first glance, an annual outage of 0.1% is already a whopping 525 minutes.

> **ⓘ Terminology - High availability, fail-safe, etc.**
> There is often some confusion when it comes to terminology. In particular, the term "fail-safe" is often wrongly used synonymously with "high availability". However, if a system is fail-safe, this does not mean that failure is impossible or improbable, but that it is built in such a way that in case of failure, it will fail in the safest possible way, that is, with minimal harm to other equipment, the environment, or to people.
> High availability, on the other hand, means that the chance of failure of the system is lower than average, or, that the system is available for longer than normal. Consequently, we will use the term "high availability" throughout this document.

Aside from the inevitable need to update the software (which in all but the most demanding environments will mean a maintenance window, that is, a planned downtime of the system, and which is not in the scope of this document), there is a plethora of unplanned things that can possibly go wrong in a computing environment, which reduce the availability of an application:

- **Software failure** - Probably the most obvious (and arguably most common) cause for a software system becoming unavailable is a crash of the software. The most obvious reasons for this are of course bugs in the software itself, and as we all know, all but the most trivial programs will inevitably have bugs. Preventing, detecting and fixing bugs (as early as possible, ideally before a customer encounters them) is perhaps the most essential task of any software R&D organization. Still, bugs *do* happen, and sometimes they will crash your entire software (i.e., the operating system process will be terminated) or the software will still be running, but will cease to function properly. The operating system itself can also have bugs that can make individual processes or the entire OS crash.

- **Hardware failure** - The hardware on which the software is running might have a defect. Mechanical components like hard disks might break down, but also solid state devices, including CPUs, chipsets, memory, SSDs, network devices, or peripheral components might fail. Sometimes failures are not permanent. For example, although rarely, data held in RAM can become corrupted (for example, a single bit can be inverted), which could cause the program to crash (and make it seem like a software failure).
- **Infrastructure failure** - The infrastructure required by an application to function and be available to its users might fail. For example, power supply or network connectivity might be interrupted, either due to human error (someone turning the wrong switch or pulling the wrong cable, an excavator used carelessly near vital power lines or network cables), or due to a large scale blackout affecting an entire region.

As it should have become clear, the **scope affected by** any such **failures** can vary:

- **Process failure** - An individual operating system process (in ARIS terminology, a single runnable) can crash, for example, due to a bug.
- **Node failure** - An entire machine can go down, for example, due to problems with the power supply or a crash of the OS.
- **Datacenter failure** - A whole data center can be affected, in particular if vital components of the infrastructure are compromised.
- **Network failure** - Even if the software and the machines itself work fine, the network connectivity between them can fail. From the perspective of the individual node, it will often be indistinguishable from the failure of the other nodes, but a network failure can produce other kinds of issues compared to a process, node or data center outage, which is why we list it separately here.

The take-away message here is: Everything can (and eventually will) fail, all the time!

The key to protecting yourself and your vital software systems against any of these failures is the introduction of redundancy.

The obvious approach to address a lot of the different kinds of failures above is to make hardware redundant. For example, you can buy servers with redundant power supplies (that might each be connected to an independent mains circuit or to an "uninterruptible power supply" device that uses battery power or even an auxiliary power generator to bridge blackouts for varying lengths of time) and with redundant network adapters that are connected to redundant network switches. You can store your vital application data redundantly on a RAID system to cover hard-drive failures, etc. The whole idea of redundancy is to reduce or ideally eliminate single points of failure in your system.

The amount of additional effort you invest in protecting your software system against outages will vary depending on how vital it is for your organization.

However, making hardware redundant has its limitations. While a lot can be done, at some point the cost will increase exponentially. Furthermore, making your hardware redundant will not help you in the case of issues with the software itself.

In a classical monolithic application, there is little that can be done at this point. Fortunately, for a software built around a modern, micro-service-based architecture like ARIS, the situation is much better. Not only can the hardware on which the software runs be made redundant, but also the individual software components themselves (the micro-services and backend services) can be made redundant by having more than one of each kind of service inside an application. If configured properly, an ARIS installation can be built so that it survives failures of various scopes:

- If you have properly configured each micro-service and backend service redundantly on a single machine, you can make your ARIS installation **resilient to a process failure** (i.e., an individual crashed runnable).
- If you properly distribute your micro-services and backend services across multiple machines, you can make your ARIS installation **resilient against the failure of an entire node** (regardless of whether it is caused by a hardware problem or a software problem).
- If you have extremely demanding availability requirements for your ARIS installation, you can configure the individual micro-services and backend services redundantly across several data centers, **making your system resilient to a data center outage**.

However, it should be noted that, while only a distributed software can potentially have higher availability than a monolithic one, the distributed software must handle failures that cannot occur in a monolithic application. For example, in a monolithic application, the individual software components communicate locally within a single operating system process, so there is no need to worry about a network outage that might temporarily make it impossible to communicate with another component. In a distributed system, where the individual components run as dedicated operating system processes that possibly reside on different machines, this communication has to happen over the network, and a sudden network issue has to be taken into account by the communicating components. A distributed application that is to become highly available must be built to handle such failures gracefully and, in particular, to prevent cascading failures that ultimately bring down the whole system. Whenever writing code that communicates with other components that might or might not be located locally, developers have to build their code to expect that communication might fail and devise ways to recover from such failures. Consequently, making software capable of being distributed introduces new levels of complexity, which can also introduce new bugs.

So for simple deployments, where high availability is not required, the additional failure modes and the potential for additional bugs will at first be a disadvantage of a distributed application over a monolithic application. On the other hand, without being able to distribute an installation across nodes or data centers (as I it is the case with a monolithic application), there is no possibility to optionally make an application highly available at all.

2.2   Making ARIS highly available

Making an application like ARIS highly available is not solely the responsibility of the developers of the software, but also of the administrators that set up an installation. Depending on the scope of errors against which you want to protect your ARIS installation, there are ways to deploy it accordingly. However, the more severe the failure that you want the system to be able to tolerate, the higher the overhead and the effort involved will be. If you don't care about high availability at all, you can stick with a single-node installation. The automatic restart of crashed components will quickly resolve minor issues and in many environments a short outage of parts of an ARIS installation can be tolerated. However, as we said above, the **key to high availability is having redundancy** built into the system: only if you have extra instances of components available, can the system survive the outage of one (or more) instance(s).

If you want ARIS to survive the outage of a single runnable, you can configure the key components redundantly on a single node (or on multiple nodes).

If you want ARIS to survive the outage of individual nodes or the outage of the network among them, you can deploy the components redundantly across multiple nodes. If an ARIS installation must remain available if an entire data center fails, you can deploy the components redundantly across data centers.

However, redundancy comes at a price: not only do you have to spend additional hardware resources that hold the redundant components, but you also need to **hold the actual data redundantly** (i.e., have several - ideally always identical - copies of the same data item), so that if a backend service instance fails, you can still access all data. Here the problem is to keep these copies (also called "replicas") consistent not only during normal operation, but also if any of the above- mentioned failures occurs.

2.2.1   Data redundancy

Let's consider that we have two instances of a backend service in which we store two copies of two data items, A and B, as shown in figure 3 below. For the purposes of this chapter, a "data item" can stand in for data at any level of granularity, e.g., a single variable value, a single tuple in a relational database, an entire table, or an entire database.

FIGURE 3: DATA REPLICATION
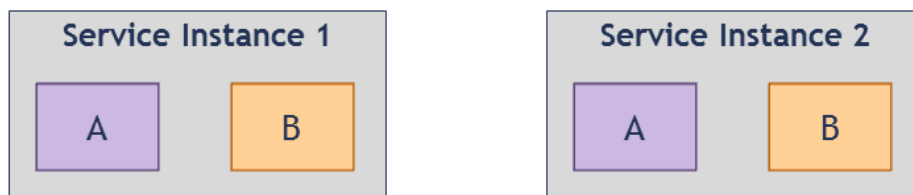
As long as the data itself doesn't change, the data from both instances can now be used to service (read) requests. The trouble starts when data is modified. There are two approaches:

- **Application-managed replica consistency**: the applications/micro-services themselves (or some component inside them) are aware of the replication and make sure that all replicas are updated.
- **Service-managed replica consistency**: the service itself (i.e., all instances together) is responsible for making sure that all replicas are updated.

2.2.1.1   Application-managed replica consistency

When the consistency of replicas is managed by the applications, this means that all application instances need to be aware of the fact that data is replicated and need to know where (i.e., in which service instance) each replica is located. When a user request (or some background job, etc.) requires that a data item be changed, the application is then responsible for updating all replicas. Continuing from our example from figure 3 above, figure 4 below shows this approach. If a write happens on data item A, the application updates both copies, leading to a new, changed version of that data item, indicated by the prime symbol ('). Only after all copies are successfully updated, is the write operation completed, and the application can proceed with serving the user request (or executing the background job.).

FIGURE 4: APPLICATION-MANAGED REPLIA CONSISTENCY
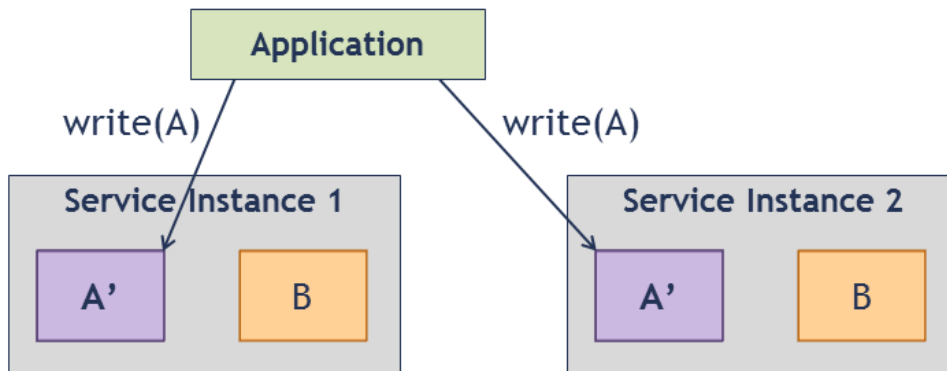
2.2.1.2 Service-managed replica consistency

The more common approach is to leave the management of replication, in particular that of updating redundant copies, to the service itself. So the application that wants to update the data item can do the write operation via **any** service instance (even via an instance that doesn't hold a replica of the particular data item), and leave it to the service instance itself to propagate the changes to the other instances and make them update their replicas. This approach is illustrated in figure 5 below.

FIGURE 5: SERVICE-MANAGED REPLICA CONSISTENCY
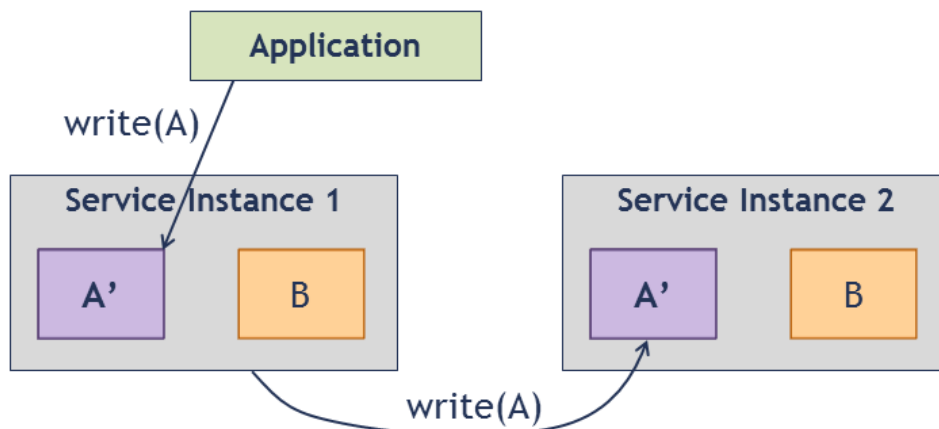
Both application-managed replica consistency and service-managed replica consistency have their pros and cons. However, the actual difficulties that you encounter when trying to maintain replicas consistently are the same in both scenarios. In the next sections, we will discuss these problems, using service-managed replica consistency in our examples. But first, we will have a look at the theoretical underpinnings of data consistency, in particular the so-called CAP theorem.

2.2.2 The CAP theorem

The CAP theorem[1], also known as Brewer's theorem, states that in a distributed system, it is impossible to provide more than two out of the following three properties[2]:

- **Consistency**: Every read operation is guaranteed to always see the most recently written value of a data item (or an error, if that is not possible).

- **Availability**: Every request receives a non-error response (but without guarantee that the response will contain the most recently written values of the involved data item).

- **Partition tolerance**: The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes.

What the this boils down to is that in the case of network partitioning, you have to choose either consistency or availability. This means you can either get the data with the risk of not seeing the most recent state of it (compromise consistency) or not get it at all (compromise availability). In the absence of any network problems, you can have both consistency and availability.

A distributed, clustered installation of traditional relational database systems will always chose consistency over availability (as only this will allow the system to provide the ACID guarantees[3] that RDBMSs always provide, i.e., **A**tomicity, **C**onsistency, **I**solation, and **D**urability), while most NoSQL systems will at least by default prefer availability, offering so-called eventual consistency (often called BASE, "**B**asically **A**vailable, **S**oft state, **E**ventual consistency", in contrast to ACID). In an eventually-consistent environment, it is not guaranteed that all redundant copies of a data item will always have the latest state, but they *eventually* will.

A few words about ACID transactions

ACID transactions[4], which are the underlying paradigm of any "traditional" database system for updating data, offer the following four properties:

Atomicity: From the perspective of clients, all changes to the data done in a single transaction are either done all at once, or not all ("all or nothing"). So either the transaction completes with a successful "commit", in which case all its changes are applied to the database, or the transaction fails or is aborted, in which case no changes at all will be applied to the database. Internally, this is usually achieved by some form of undo mechanism. For example, the database system will write transaction log files that allow it to undo any change that has already been applied in case of an error or abort by the client or redo any changes of a committed transaction that have not yet been persisted.

Consistency: The changes done in a transaction will bring the content of the database from

---

[1] Gilbert, Lynch: Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services, ACM SIGACT, Volume 33 Issue 2, June 2002,Pages 51-59 (PDF)

[2] Wikipedia: The CAP theorem https://en.wikipedia.org/wiki/CAP_theorem

[3] Wikipedia: ACID https://en.wikipedia.org/wiki/ACID

[4] Wikipedia: ACID https://en.wikipedia.org/wiki/ACID

one consistent state to a new consistent state. Note that here consistency means "being consistent from the perspective of the database system's data model", i.e., all constraints defined by the data model (in the case of the relational model, such constraints are in particular referential integrity, the uniqueness of primary keys, etc.) are maintained. This is in general handled by checking the consistency constraints at the end of the transaction and aborting the transaction in case of a constraint violation (or by actively performing the necessary operations to make the database consistent again, e.g., by cascading deletes).

Isolation: Different transactions working concurrently on the same set of data will see the database state as if they were executed strictly sequentially (in any possible order). A simple approach to achieve this guarantee is by using a locking approach with read and write locks: before a data item can be read by an application, a read lock has to be set; before a data item can be changed, a write lock has to be set. Locks, once they are set, are then held until the end of the transaction (i.e., until it is committed or aborted). When a transaction tries to set a lock on a data item and no other transaction is currently holding a lock on it, the lock operation will complete successfully, the transaction will obtain the lock and can continue working with the data item. Different read locks on the same data item acquired by different transactions are "compatible" and can coexist (because concurrent reading is possible without any conflicts). Write locks are not compatible with read locks or other write locks. So if a transaction attempts to obtain a write lock in order to change a data item, but other transactions have already set a read lock, the write lock cannot be obtained. This situation is shown in the figure below.

A simple example of locking-based concurrency control
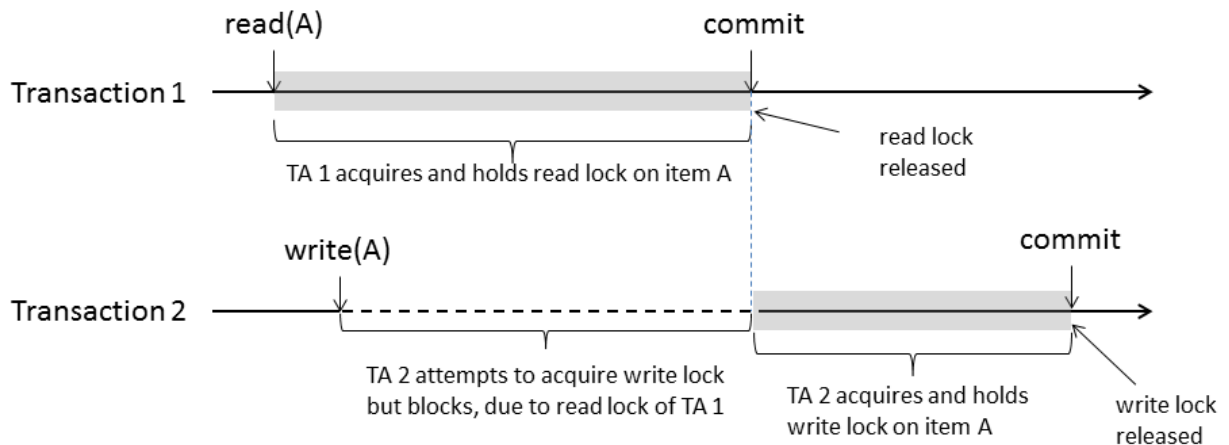
Transaction 1 (TA 1) performs a read on data item A. In order to be allowed to do this, it acquires and holds a read lock, which is held until it commits. Briefly after TA 1 acquires the read lock on A, a second transaction TA 2 attempts to change the same data item, for which it needs to acquire a write lock. Due to the read lock already held by TA 1, the attempt to get the write lock is blocked, so TA 2 is suspended. Only after TA 1 commits and releases its read lock, is the blocked attempt of TA 2 to get the write lock unblocked, and TA 2 can obtain the write lock and modify A, itself releasing the write lock when it commits. Similarly, if a transaction tries to set a read lock but a write lock is already set, the reading transaction is

blocked until the writing transaction completes (or aborts) and releases the write lock.

In effect, any transaction will see the database in a state as if it was running sequentially with all other applications (i.e., one transaction after the other), even though the transactions can run concurrently to some degree (until they have to wait for getting a lock on a data item that is currently locked by another transaction). Much more sophisticated locking mechanisms, or isolation mechanisms that do not require locking (so-called **optimistic concurrency control** mechanisms, e.g., multi-version concurrency) exist, which allow higher degrees of concurrency, but all offer the same level of isolation, i.e., the database state will appear to each transaction as if it was running exclusively on the database.

Durability: The changes done in a transaction that successfully committed stay that way even in the case of errors, power failures, etc. This property is guaranteed by making sure that changes are persisted to the data storage (which is slow) or by making sure that the change is written to the transaction log (which is written sequentially, which can be done faster), so that they can be reapplied in case of a system crash.

2.2.3 Maintaining replica consistency

2.2.3.1 Strict consistency

If you want to preserve strict consistency of data replicas, all replicas have to be updated synchronously within the write request duration, which means that the duration of an application's write requests will increase. Even if the write operations are done in parallel, the **slowest write will be the limiting factor of performance**. Further, guaranteeing that all replicas are updated consistently incurs significant overheads (e.g., a variant of the two-phase-commit protocol, 2PC[5]). This approach is shown in figure 6 below.

FIGURE 6: SYNCHRONOUS, STRONGLY-CONSISTENT UPDATE OF REPLICAS
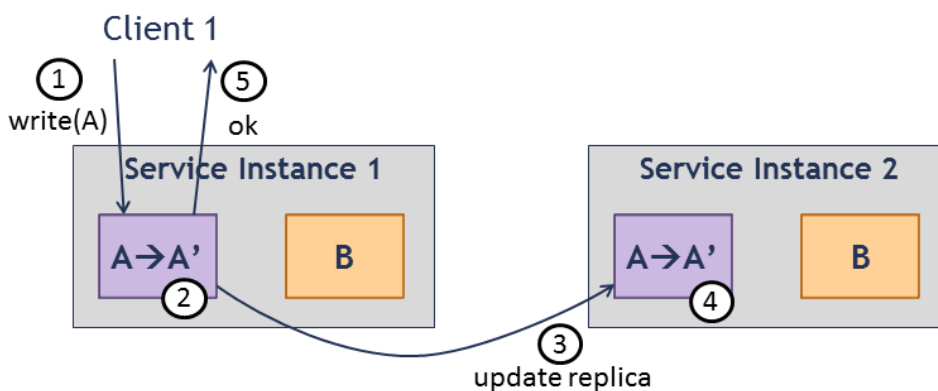
The client write request reaches a service instance (1), which updates its local replica of the data item (2), then updates the remote replica (3 and 4), and only then returns a "successful" response to the client. If an error occurs while updating the remote replica (e.g., if service instance 2 becomes unavailable), the change to the data item is rolled back and an error is reported to the client instead. This ensures that either both replicas are updated or none of them is. Further, concurrency control mechanisms (as the ones described when we discussed ACID transactions above) ensure that other clients that try to access the data replica in either instance during the write operation by client 1 will either see the old state of A and block the write operation of client 1 until their reads have been completed, or will themselves become blocked until the write operation of client 1 has completed successfully (after which they will see the modified version A') or failed (in which case they will see the original version of the data).

Things start to get really difficult if you consider various error cases: In particular, what happens if a service instance holding a replica of a data item that is to be updated is currently unavailable? Should the application wait for it to become available? Even in the best case scenario, that could take minutes, or the instance might never come back, either causing the write operation to hang indefinitely, or run into a timeout and return an error. So **if you favor consistency, you sacrifice availability**, just as the CAP theorem states.

---

[5] Two-phase commit protocol https://en.wikipedia.org/wiki/Two-phase_commit_protocol

2.2.3.2   Eventual consistency

Another option is to live with the situation that replicas are not always in a consistent state, at least for some time. Figure 7 below shows how a hypothetical service could handle this: The client write request reaches a service instance (1), which updates its local replica of the data item (2), and then immediately returns a "successful" response to the client. The replica of data item A located in service instance 2 is not yet updated. A second client could now read data item A. If it accessed service instance 2, which still has the replica that hasn't been updated yet, client 2 would still see the old value of A (4 and 5). Only after service instance 1 has **eventually** informed service instance 2 that it should update its replica (6 and 7), are all replicas again in a consistent state and only then will all clients that access the data item through any service instance see the same value.

FIGURE 7: DEFERRED, EVENTUALLY-CONSISTENT UPDATE OF REPLICA
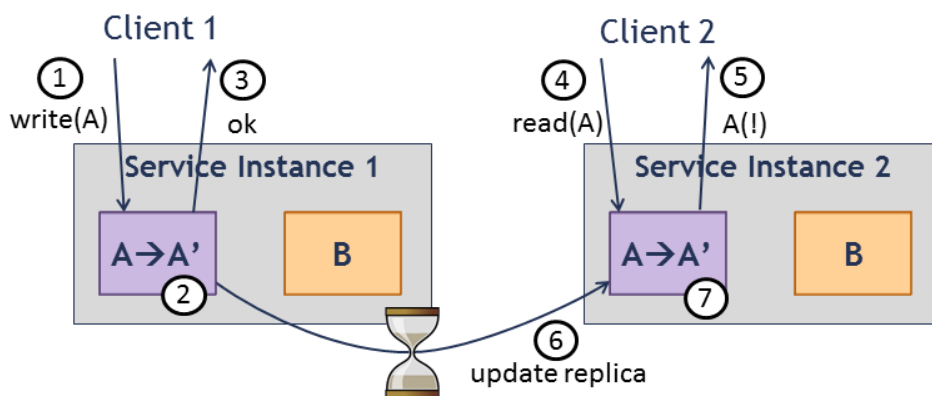
An obvious advantage of this approach - aside from replica updates happening asynchronously to the actual write operations and thus not slowing down client requests - is that it is now possible to still serve write requests even if a service instance holding a replica is currently unavailable. However, now applications might see "stale" data. Depending on the actual application requirements, in particular for applications where a large part of the clients are only browsing the data, this might be acceptable. If, for example, some clients of a webpage see the old version of the page for a short time, this can often be tolerated.

However, in an eventual consistency model, keeping the replicas consistent is much more challenging. Consider the situation shown in figure 8 below:

FIGURE 8: CHALLENGES WHEN UPDATING REPLICA IN AN EVENTUALLY CONSISTENT SERVICE
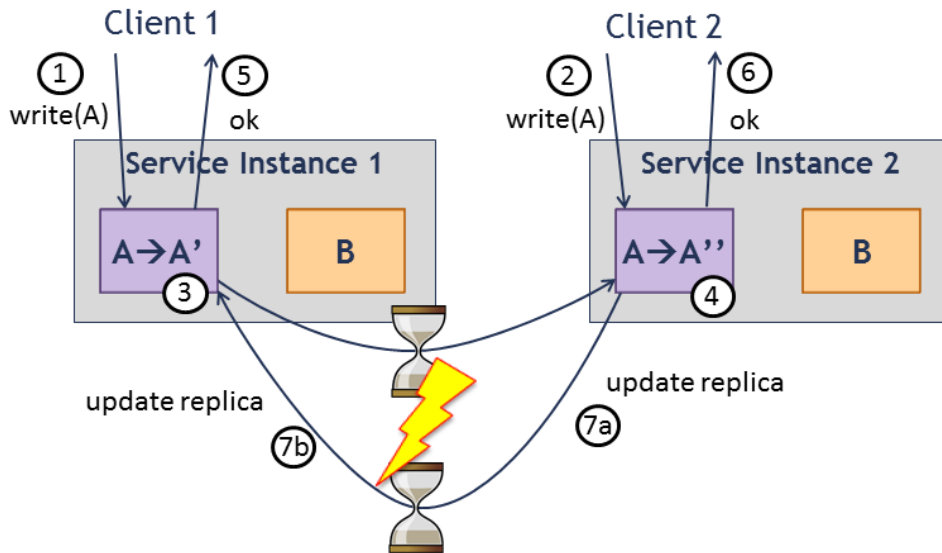
Here two clients make a change to the same data item, but through different service instances. Client 1 changes the replica of data item A in service instance 1 (1, 3, and 5) to state A', while concurrently, client 2 changes the replica in service instance 2 (2, 4, and 6) from A to A''. When the service instances try to update the respective other replicas, a conflict occurs because it is not clear which changed state of data item A, A' or A'', is the correct one. You must try to avoid this situation because it is not possible to resolve the resulting conflict automatically.

A common approach is a simple master-slave replication scheme. Here either an entire service instance or one explicitly selected replica of each data item is declared the "master", through which all write operations happen. In figure 9 below, we indicate which replica is the master replica by underlining it. Here service instance 1 holds the master replica of data item A and a "slave" or secondary replica of item B, while service instance 2 holds the master replica of data item B and a secondary replica of data item A.

FIGURE 9: MASTER AND SLAVE REPLICA
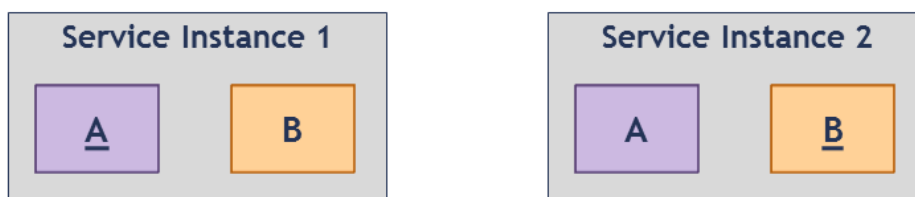
Write operations to any data item are now managed by the instance holding the master replica, which is then responsible for propagating the change to all other instances. The instance holding the master replica of a data item is a possible bottleneck for writes to it. However, because the master replicas of multiple data items can be distributed across all instances, the overall write load over all data items can also be spread out.

The next problem is how to decide which replica will be the master, and how to make sure that there is exactly one master at any given time I If there is no master, no writes can be performed, and if there are multiple masters, a problem like the one shown in figure 7 above could still occur. While the details of the actual mechanism with which this is achieved go beyond the scope of this document, the general idea is that an "election process" for the master is performed. However, even with this approach, problems can occur. Consider the situation shown in figure 10 below:

FIGURE 10: NETWORK PARTITIONING
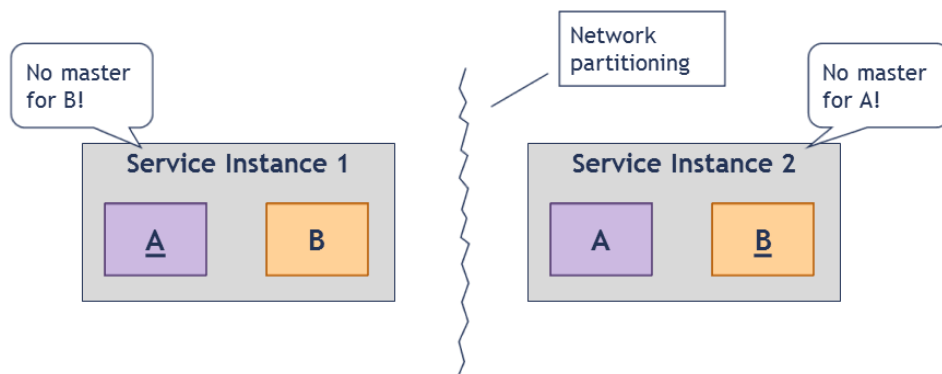
Here, the communication between the two service instances is interrupted, a common situation in distributed computing environments. From the perspective of each instance, it might appear that the other instance has failed, even though the problem is just the network between them. Instances 1 and 2 could now notice that they do not have access to a master of data item B or A, respectively. Without precautions, the instances might start an election process for a new master, and because they are the only instances voting, each instance could conclude that it is now master for both data items. Observe that we now again have two masters for each data item, as shown in figure 11 below. This dreaded situation is called the **split brain problem**[6].

---

[6] Wikipedia: Split-brain (computing) https://en.wikipedia.org/wiki/Split-brain_(computing)

FIGURE 11: SPLIT-BRAIN PROBLEM - MULTIPLE MASTERS IN THE CASE OF A NETWORK PARTITIONING
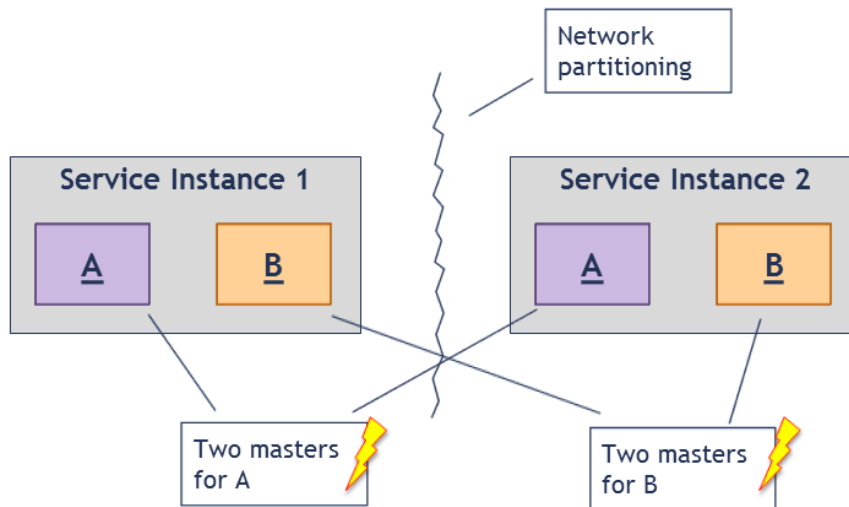
When we are in this split-brain situation, clients on each side of the partitioning can modify the same data items independently, as shown in figure 12.

FIGURE 12: SPLIT-BRAIN IN CASE OF MULTIPLE MASTERS DURING A NETWORK PARTITIONING
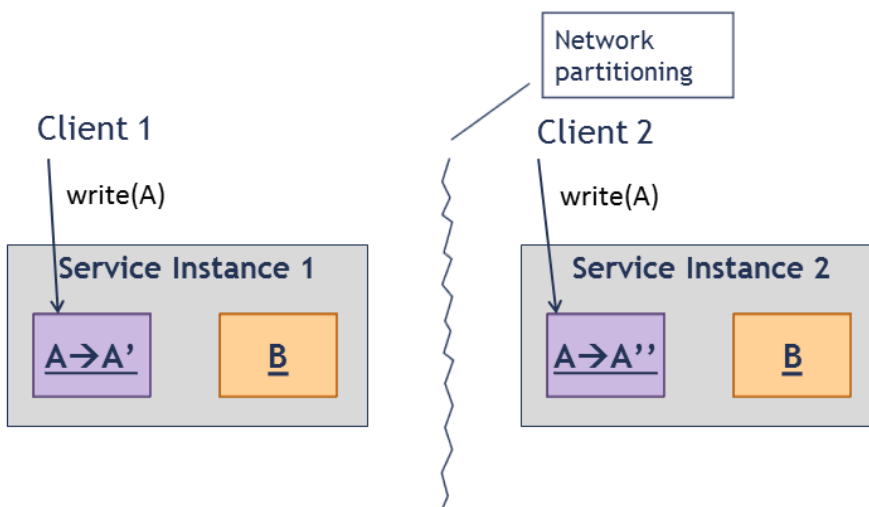
The state of the data items on both sides of the network partitioning could now diverge from each other. After the network partitioning is resolved, the issue will become apparent, when each master notices the competing master from the other side of the former partitioning. In general, the different states of the data items cannot be resolved automatically (at least not satisfactorily), requiring human interaction to resolve the conflicts.

FIGURE 13: CONFLICT AFTER THE NETWORK PARTITIONING IS RESOLVED
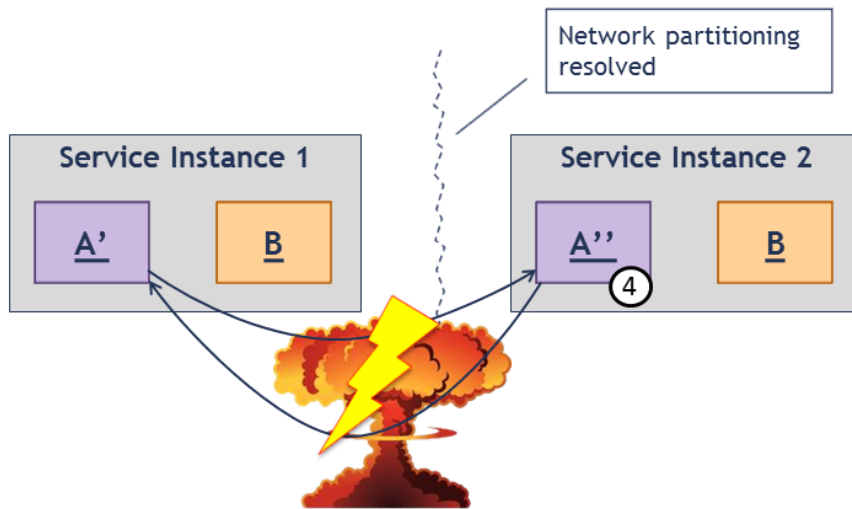
A common approach to avoid the election of multiple masters is to require the instances on each side of the network partitioning to obtain a quorum, i.e., require more than 50% of the voting instances (>50% **not** ≥50%) to be available before they can start the voting process for a new master. That way, only the bigger partition (i.e., the one having more instances in it) can elect a master and continue to operate, while the smaller partition will either not allow any operations at all or at least not allow any write operations. In our example scenario that we sketched so far we only had two instances of our service.

Consequently, in a network partitioning situation like the one depicted above, neither partition would obtain the necessary number of voting instances for a quorum, and thus the service would become unavailable on both sides. There are a few approaches to avoiding such ties: the most common one is to make sure that there is always an odd number of instances in the total system, so that no matter how the instances are partitioned, there is one side with a majority. Another "tie breaker" is the assignment of weights to instances, so that the side with at least 50% of the total weight can continue to operate. For example, in our two-instance scenario, we could have assigned a weight of 51% to service instance 1, and 49% to instance 2.

Notice how all the different problem scenarios discussed above are all a manifestation of what is stated by the CAP theorem: In the presence of network partitioning, you can either have reduced consistency (by using a consistency model and/or risking a split-brain situation) or reduced availability (by using a strict mechanism of maintaining replica consistency or by using a quorum approach, where the smaller partition will not be able to function).

3   Preparation steps

3.1   Preparing a fresh installation

Before you can do the actual installation of a distributed ARIS installation, you must take several preparation steps, which are described here. A prerequisite is that you have the required number of virtual or physical worker machines for the scenario you intend to set up, and that you have administrative access to the worker machines.
The number of machines depends on the system environment, the expected load (for example, the number of current modelers and viewers), the amount of application data (for example, the number and size of ARIS databases, the amount of documents, and the total number of users).

However, for a number or reasons that we are going to learn about over the course of this document, a minimum requirement for a distributed system that also offers some degree of high availability is to have at least three nodes. For the example scenario that we are going to use as a running example throughout this document, we assume that we have three nodes available. In our example commands, we further assume that the hostnames of these three nodes are arisworker1.company.com, arisworker2.company.com, and arisworker3.company.com, to which we will briefly refer to as "node 1" (or "n1"), "node 2" (or "n2"), and "node 3" (or "n3"), respectively, as illustrated in the figure below.

FIGURE 14
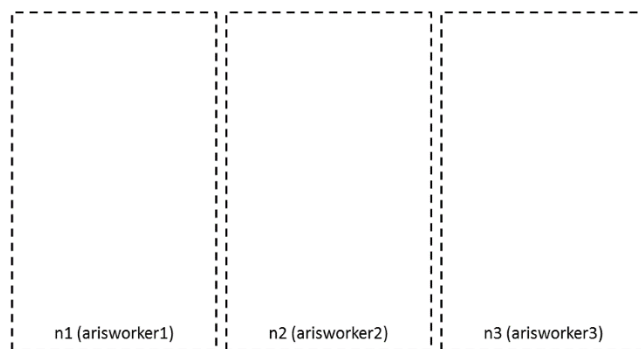
Over the course of this document, we will extend this basic diagram so that it contains all the components that we already installed.

In many less demanding or run-of-the-mill scenarios where customers are interested in a highly available ARIS installation, the example scenario we are going to develop in this document will probably work just fine. However, for foreseeably more demanding requirements the best approach is a staged ramp-up of the introduction of ARIS: After starting with an initial system configuration and initial set of users, the system is then carefully monitored for overall performance. As more and more users are allowed access and more data is put into the system, the system is then reconfigured and/or expanded accordingly (i.e., scaled out, by adding more machines and runnables) depending on where performance bottlenecks show up.

3.2 Install ARIS Cloud Controller on your admin machine

First, you must install ARIS Cloud Controller on your "admin machine", i.e., the machine from which you will perform the installation. You can do so with the installers provided on the ARIS DVD.

On Windows, you can install ACC with the ARIS Agent setup (located at <DVD_Root>\Setups\Windows\ARIS_Agent). The setup program is straightforward to use, you only need to choose an installation directory.
On Linux, you can use the installation package for your respective Linux distribution, which you can find at <DVD_Root>\Setups\Linux_RedHat\ARIS_Cloud_Controller or <DVD_Root>\Setups\Linux_SUSE\ARIS_Cloud_Controller. You can install these packages with the usual commands of your distributions package manager (in short: "rpm -i <package>" for RPM-based distributions, "dpkg -i <package>" for Debian-based distributions).

⚠ There is currently no separate installer available to install only ACC on Windows; it is always installed together with the agent via the ARIS Agent setup. If you do not want to install the agent on your admin machine, you can install the agent (incl. ACC) via setup, then stop the Windows service ("ARIS Agent"), and set its start type to "manual".

You can later invoke ACC from a command window/shell using <pathToAccInstallationDirectory>\acc.bat (on Windows) or <pathToAccInstallationDirectory>\acc.sh (on Linux).
On Windows, the ACC batch file is in <installationDirectory>\server\acc (where <installationDirectory> is the path you chose in the agent setup directory. On Linux distributions, the ACC bash script file is usually in /usr/bin/acc (but that might differ depending on distribution).
On Windows, you might want to add the directory containing acc.bat to your PATH environment variable, so that you can invoke ACC from a command prompt positioned in any directory. On Linux, the package installer usually adds acc.sh to the path.
If you do not do that, you will later have to specify the full path to acc.bat when running ACC. In all the following steps we assume that you changed the PATH environment variable, so will call ACC without specifying the full path to it.

3.3   Create a project directory

On the admin machine from which you will be using ACC, **create a project directory** (e.g., c:\aris_distributed) to which you will copy all the various config files we create in the next sections.

> ⓘ If you do not want to change the PATH of your Windows system, you can simply use the directory containing acc.bat as your project dir and put the files there.

3.4   Install the ARIS Agent on all nodes

Next, install ARIS Agent on all nodes that you want to use for the distributed installation by using the installers/packages for the respective operating systems.
On Windows, you can install the agent by using the agent setup located at
<DVD_Root>\Setups\Windows\ARIS_Agent. The setup program is straightforward to use. You must choose only an installation directory.
On Linux, you can use the installation package for your Linux distribution located
at,at <DVD_Root>\Setups\Linux_RedHat\ARIS_Agent
or <DVD_Root>\Setups\Linux_SUSE\ARIS_Agent, respectively.
You can install these packages by running the usual commands of your distribution's package manager. In short, run "rpm -i <package>" for RPM-based distributions or "dpkg -i <package>" for Debian-based distributions. For more details, see the "Server Installation Guide - Linux" on the ARIS DVD.

3.5   Controlling multiple nodes with a single ACC instance

While many users are familiar with using ACC to control a simple single-node installation, a single ACC instance is capable of working with multiple agents of a distributed installation. This section describes the basic commands used to register nodes with ACC.

3.5.1   A look at the common ACC command line switches

To start ACC in a single node installation, you pass the hostname, username, password, the path to the generated.apptypes.cfg file (more on that file later), and optionally the agent's port to the ACC script file. via the command line, following this general structure:

```
acc.{bat|sh} –h <hostname> –p <agentPort> –u <agentUsername> –pwd
<agentPassword> –c <pathToGeneratedAppTypesCfgFile>
```

(The ACC script file is named acc.bat on Windows and is located at <installationDirectory>\server\acc. The ACC script file is named acc.sh on Linux and is located at /usr/bin/acc.)

For example, in a single-node Windows installation in the standard installation directory, where you did not change the agent username and password, you start ACC as follows:

```
acc.bat –h localhost –p 14000 –u Clous –pwd g3h31m –c
c:\SoftwareAG\ARIS10\server\acc\generated.apptypes.cfg
```

Note that you can leave out the -p command line switch if the agent is running on its default port, which is 14000 in ARIS 10 (and used to be 9001 in ARIS 9.x).

This is also the command line used in the Windows start menu shortcut ("ARIS 10.0 Cloud Controller") that is created by the setup in a single-node installation, which you can inspect via the Properties context menu entry:

FIGURE 15
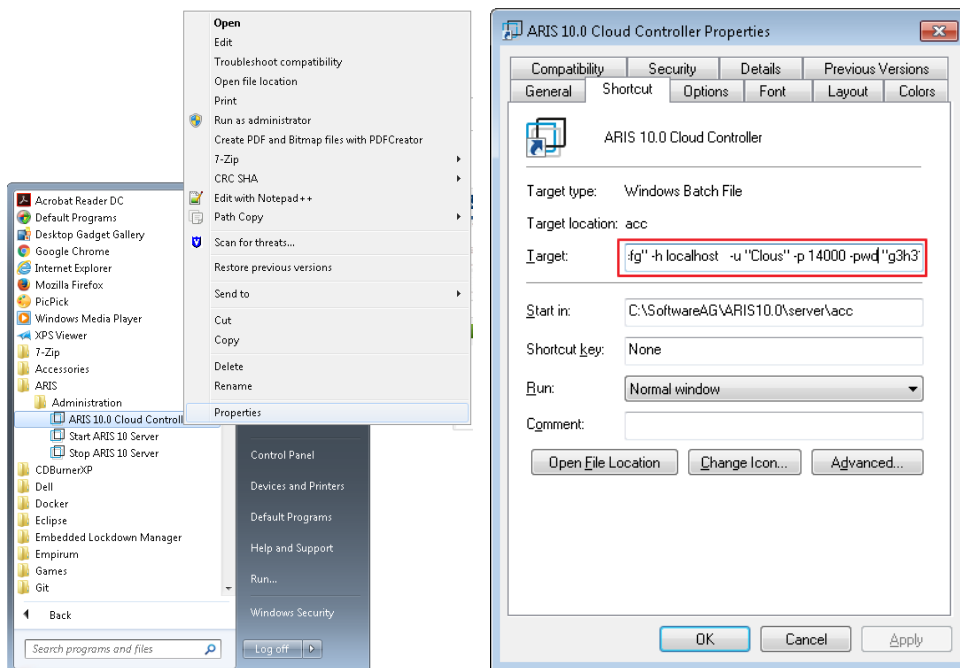
For the next few steps, it is recommended that you use any single-node installation of ARIS 10 at your disposal and start ACC with this command line. It will give you the familiar prompt:
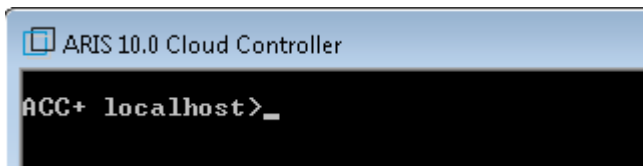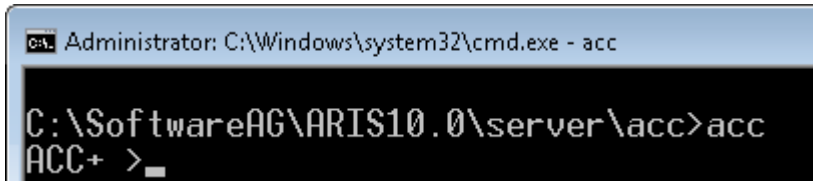
You can now start issuing the well-known commands (e.g., "list" to display all currently configured runnables).

3.5.2 A different way of starting ACC
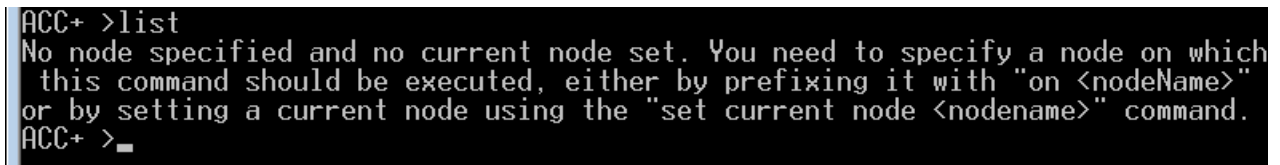
However, you can also start ACC using the script file without specifying any command line options. Open a command prompt in the directory containing the ACC script file (or in any directory when on Linux) and enter "acc".

This will give you a prompt like this:
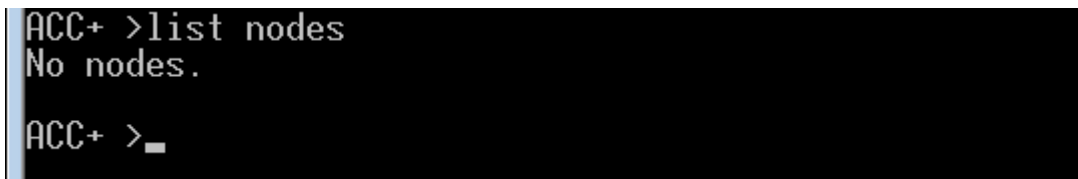
```
Administrator: C:\Windows\system32\cmd.exe - acc

C:\SoftwareAG\ARIS10.0\server\acc>acc
ACC+ >_
```

Notice that the prompt does not say localhost. If you try to run a command (try "list"), ACC will return an error:

```
ACC+ >list
No node specified and no current node set. You need to specify a node on which
 this command should be executed, either by prefixing it with "on <nodeName>"
or by setting a current node using the "set current node <nodename>" command.
ACC+ >_
```

This is ACC's way of telling you that it does not know to which node to send a command. actually, at this point ACC does not know about **any** nodes. You can check this with the "list nodes" command, which lists all nodes currently known to ACC:

```
ACC+ >list nodes
No nodes.

ACC+ >_
```

Without our usual command line options, ACC does not know to which agent to talk. Instead of going back to using the command line switches, you can also use a command to make a new agent known to ACC, the **add node** command.

3.5.3   The "add node" command

The basic structure of the command is as follows (you can also get a help text displayed in ACC by typing "help add node"):

```
add node <logicalNodeName> <ipAddressOrHostname> [:<agentPort>]
<agentUsername> <agentPassword>
```

You will recognize the familiar parameters that we normally pass to ACC on the command line, i.e., the hostname or IP address of the machine (or often just "localhost"), the optional agent port, and the username and password of the agent user.

However, you will also notice an additional parameter, <logicalNodeName>. For this parameter you can specify a shorthand name with which you can later refer to the node in ACC commands. For starters, let's just add the agent on localhost with the command, and let's name it "localhost" as well (In general, it is not a good idea to use the hostname also as the logical node name… we do this in this section for didactic purposes).

```
add node localhost localhost Clous g3h31m
```

(Note how the optional port was omitted, because the agent installed as part of an ARIS 10 single-node installation will be listening for requests on the default port.)

If the command executed successfully, the output looks like this and you can use the "list nodes" command to check that your local agent was registered with ACC:

```
ACC+ >add node localhost localhost Clous g3h31m
Agent at localhost:14000 added successfully with logical node name localhost.
ACC+ >list nodes
One node:
localhost : localhost (14000) OK

ACC+ >_
```

you remove a node from the ACC's list of registered nodes using the "remove node" command. The basic syntax is as follows:

```
remove node <logicalNodeName>
```

To remove our node localhost from ACC, type:

```
remove node localhost
```

After trying "remove node", re-add the node by running the previous "add node" command again (on Windows, you can scroll through the previous commands using the arrow-up and arrow-down keys).

3.5.4   Qualifying commands with the node name

So by now, we should be able to use our usual commands, right? Let's try "list" for starters:

```
ACC+ >list
No node specified and no current node set. You need to specify a node on which
 this command should be executed, either by prefixing it with "on <nodeName>"
or by setting a current node using the "set current node <nodename>" command.
ACC+ >_
```

The command returns the same error as before.

The error message already points out the problem and the solution: while it knows about a single agent, it still insists that you explicitly tell it to which agent to talk. There are two ways of doing this.

3.5.4.1   The "on <nodename>" clause

All commands that pertain to a single runnable, or to a group of runnables on a single node, can be prefixed with the "on <logicalNodeName>" clause, in which you specify the logical node name of the node to which to send the command, for example:

```
on localhost list
```

Let's try this out now:

```
ACC+ >on localhost list
Node localhost - 14 installed runnables.
ID              State       Version   Type
zoo_m           STOPPED     10.0.0.2  com.aris.runnables.zookeeper-run-prod
postgres_m      STOPPED     10.0.0.2  com.aris.runnables.PostgreSQL-run-prod
cloudsearch_m   STOPPED     10.0.0.2  com.aris.cip.y-cloudsearch-run-prod
elastic_m       STOPPED     10.0.0.2  com.aris.runnables.elasticsearch-run-prod
umcadmin_m      STOPPED     10.0.0.2  com.aris.umcadmin.y-umcadmin-run-prod
adsadmin_m      STOPPED     10.0.0.2  com.aris.adsadmin.y-adsadmin-run-prod
abs_m           STOPPED     10.0.0.2  com.aris.modeling.components.y-server-run-prod
octopus_m       STOPPED     10.0.0.2  com.aris.octopus.y-octopus_server-run-prod
apg_m           STOPPED     10.0.0.2  com.aris.age.age-run-prod
copernicus_m    STOPPED     10.0.0.2  com.aris.copernicus.copernicus-portal-server-run-prod
ecp_m           STOPPED     10.0.0.2  com.aris.runnables.ecp-run-prod
simulation_m    STOPPED     10.0.0.2  com.aris.modeling.components.simulationserver.y-simuserver-run-prod
loadbalancer_m  STOPPED     10.0.0.2  com.aris.runnables.httpd.httpd-run-prod
dashboarding_m  DEACTIVATED 10.0.0.2  com.aris.dashboarding.dashboarding-run-prod
```

As you can see, ACC now knows that it should return a list of all runnables on the node that we registered with the logical node name "localhost". However, prefixing each command with "on <logicalNodeName>" adds significantly to the amount of typing you have to do, in particular when you are working with a specific node or have a single node registered in ACC. For these cases, there is a better option, also mentioned in the error message above.

3.5.4.2   Setting a "current node"

As an alternative to specifying the node to use with each command, you can tell ACC that all subsequent commands that pertain to a single runnable, or to a group of runnables on a single node, should be sent to a specific node by default by using the "set current node" command. The "set current node" command has the following syntax:

```
set current node <logicalNodeName>
```

In our example, where we registered the agent with the logical node name "localhost", the command would be:

```
set current node localhost
```

Try this:

```
ACC+ >set current node localhost
Node localhost is set as current node.
ACC+ localhost>_
```

Observe how the ACC prompt changed and now shows the logical node name.

Now you can again run your "list" command without having to prefix it with "on <logicalNodeName>".

> ℹ️ After using the "set current node" command on an ACC that was started without any command line options, and where we added a single agent using the "add node" and "set this node" commands as the current node, the prompt will now look exactly as it does if the ACC is started by using the -h, -u, and -pwd command line switches on "localhost".
> Basically, these command line switches are a shortcut for the commands
>
> ```
> add node <hostNameGivenOnAccCommandLine> <hostNameGivenOnAccCommandLine>
> <userNameGivenOnAccCommandLine> <passwordGivenOnAccCommandLine>
> ```
>
> For example, when starting ACC from a OS command prompt like this
>
> ```
> acc.bat -h localhost -u Clous -pwd g3h31m [...]
> ```
>
> as it is done in the ACC shortcut created by the setup in a single node installation, under the hood the following commands are executed:
>
> ```
> add node localhost localhost Clous g3h31m
> set current node localhost
> ```
>
> The intention is to make the most common use case - that of using ACC to control a simple single-node installation - as convenient as possible -, ideally without the casual ACC user even knowing that ACC can manage multiple nodes.

⚠ While in most cases the hostname given with "-h" is also used directly as a logical node name, there is some conversion being done to turn the hostname into a valid node name. In particular, when an IP address is given with the "-h" switch instead of a hostname, an attempt is made to find a proper hostname for that IP (by using a reverse DNS lookup). If a hostname cannot be found, the dots (".") (for IPv4 addresses) or colons (":") (for IPv6 addresses) are replaced with the underscore ("_") character, as neither dots nor colons are allowed in logical node names.

⚠ If you have another ARIS installation at your disposal, you can now add its agent to your ACC by issuing a corresponding "add node" command. However, unless you really know what you are doing, **never add nodes belonging to a different installation to the same ACC**. While the commands for which you have explicitly specified the node and the runnable that they apply to will work fine, other commands will be implicitly applied on all runnables (usually of a certain type) across all nodes, which will lead to unforeseeable behavior of these commands.

ACC has certain indications to find out that nodes belonging to different installations have been added. If ACC does notice such a situation, you will see a warning like the following:

```
ACC+ localhost>add node n2 ▨▨▨▨ Clous g3h31m
Agent at ▨▨▨▨:14000 added successfully with logical node name n2.
There was a warning:
There are 2 Zookeeper ensembles across the node(s) n2  and  localhost.

ACC+ localhost>_
```

Basically, ACC noticed that across the nodes you registered there are multiple Zookeeper instances, which, however, are not connected to form a single Zookeeper cluster (we will learn about clustering Zookeeper in chapter 4) - this is a strong indication that either someone made a mistake when configuring Zookeeper or that nodes that belong to different installations were added to ACC.

3.6 Adding our worker nodes to ACC

By using what we learned about the "add node" command and controlling multiple nodes with a single ACC, we can now use multiple "add node" commands to make a single ACC instance know about all the agents we just installed on our worker nodes.
Let's try this now: start the ACC you installed on your admin machine, without specifying any command line options.
Type an "add node" command for each of your worker nodes. Remember the basic syntax of the "add node" command introduced above:

```
add node <logicalNodeName> <ipAddressOrHostname> [:<agentPort>]
<agentUsername> <agentPassword>
```

Replace <logicalNodeName> with the logical node name you want to assign. Later you will use this name to refer to the node in ACC commands. You should choose a reasonably short but expressive name. If you do not have clearly defined roles for certain nodes, you can simply number them, for example "n1", "n2", and so on….). You need to specify the agent port (separated from the hostname by a colon ":") only if you want to run the agent on a different port than the default (14000 for ARIS 10, 9001 for ARIS 9).

ⓘ **Why we introduce logical node names**

So why do we actually introduce the logical node names and not just later refer to the nodes by their hostname or IP address?

First, hostnames or IP addresses are often long and easily mistyped. Moreover, assigning a logical node name and using this name in all commands allows you to easily exchange the actual nodes on which you run your commands - all you need to do is update the hostnames or IP addresses in your "add node" commands. All commands you created (in particular the "template" file we are going to create which contains the "configure" commands that decide which runnables are put on which node) can then be easily applied to a different environment (the same number of nodes and roughly the same hardware sizing presupposed), i.e., your scripts become portable.

Assuming that the hostnames of our worker nodes are named arisworker1.company.com, arisworker2.company.com, and arisworker3.company.com, and we just named those nodes n1, n2, and n3, the corresponding add node commands are as follows:

```
add node n1 arisworker1.company.com Clous g3h31m
add node n2 arisworker2.company.com Clous g3h31m
add node n3 arisworker3.company.com Clous g3h31m
```

If everything works fine, the output similar to this:

```
C:\aris_distributed>acc
ACC+ >add node n1 arisworker1.company.com Clous g3h31m
Agent at arisworker1.company.com:14000 added successfully with logical node name n1.
ACC+ >add node n2 arisworker2.company.com Clous g3h31m
Agent at arisworker2.company.com:14000 added successfully with logical node name n2.
ACC+ >add node n3 arisworker3.company.com Clous g3h31m
Agent at arisworker3.company.com:14000 added successfully with logical node name n3.
ACC+ >_
```

If you have problems adding a node to ACC, this might be caused by the agent not having been installed properly, or because the agent is not running.

You can now run the "list nodes" command to see all nodes registered in ACC:

```
ACC+ >list nodes
3 nodes:
n1 : arisworker1.company.com (14000) OK
n2 : arisworker2.company.com (14000) OK
n3 : arisworker3.company.com (14000) OK

ACC+ >_
```

3.7   The "on all nodes" clause

You can execute a number of commands on all nodes currently known to the ACC by prefixing the commands with the "on all nodes" clause.

For example, you can use "on all nodes" with the "list" command:

```
ACC+ >on all nodes list

Node n1 - NO installed runnables.

Node n2 - NO installed runnables.

Node n3 - NO installed runnables.

ACC+ >
```

Or you can use the "on all nodes" clause with "get" and "set" commands that allow you to read resp. change the value of an agent configuration parameter. For example, type

```
on all nodes get rest.port
```

to get the value of the "rest.port" agent config parameter (which is the network port the agent's REST interface is listening on for requests and will be 14000 by default):

```
ACC+ >on all nodes get rest.port

On node n1 the value for "rest.port" is : "14000"

On node n2 the value for "rest.port" is : "14000"

On node n3 the value for "rest.port" is : "14000"

ACC+ >
```
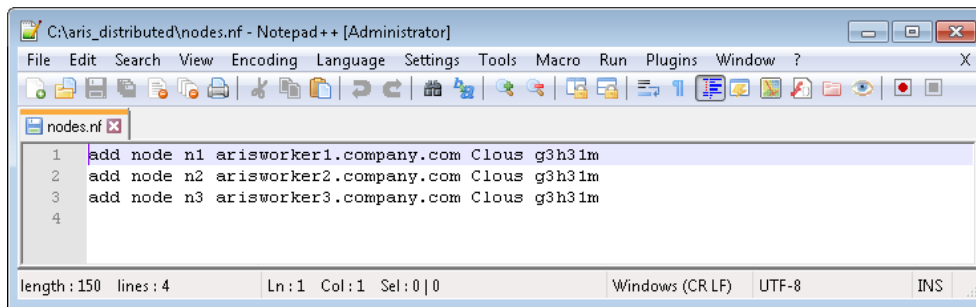
Other commands that can be used with "on all nodes" are "startall", "stopall", "killall", "updateall", and "deconfigureall". We will learn more about most of these commands in the chapter about administration of a distributed installation.

3.8 Creating a node file

We can now control our three worker nodes with a single ACC instance. However, the regular ACC console does not persist nodes added to it in a previous session, and quite obviously, manually entering the "add node" commands every time we start ACC will be tedious. Fortunately, ACC offers a better way to tell it about the nodes that are part of an installation, a so-called "node file".

A node file is a text file containing an "add node" command for each of the nodes that are to be part of your distributed installation. For your node file you can choose any name you want, but for the remainder of this document we will assume that you named it "nodes.nf" and created it in your project directory. Just add the three "add node" commands from above to that file and save it:

FIGURE 16

3.9   Using the node file

Now it is time to start an ACC and make it use our node file. You can pass the path to the node file by using the "-n" command line switch, as follows:

```
acc -n <pathToNodeFile>
```

For example, assuming you created your node file in the directory "c:\aris_distributed" and named it "nodes.nf", you would run it with

```
acc -n c:\aris distributed\nodes.nf
```

Try this out now. If everything works, the output should look like this:

```
C:\SoftwareAG\ARIS10.0\server\acc>acc -n c:\aris_distributed\nodes.nf
Executing command "add node n1 arisworker1.company.com Clous g3h31m" in line 1 of file c:\aris_distributed\nodes.nf
Agent at arisworker1.company.com:14000 added successfully with logical node name n1.
Executing command "add node n2 arisworker2.company.com Clous g3h31m" in line 2 of file c:\aris_distributed\nodes.nf
Agent at arisworker2.company.com:14000 added successfully with logical node name n2.
Executing command "add node n3 arisworker3.company.com Clous g3h31m" in line 3 of file c:\aris_distributed\nodes.nf
Agent at arisworker3.company.com:14000 added successfully with logical node name n3.
Validation of the nodes n1, n2 and n3 completed without errors.
ACC+ >
```

If you added ACC to the PATH environment variable, it is best to simply change to your project directory and start ACC from there, so you can leave out the path in the ACC command line, for example:

```
c:\aris distributed>acc -n nodes.nf
```

If everything works as expected, ACC should start up and add the nodes listed in your node file without giving any errors, and leave you with a prompt without a current node set (i.e., "ACC+ >"):

```
C:\aris_distributed>acc -n nodes.nf
Executing command "add node n1 arisworker1.company.com Clous g3h31m" in line 1 of file c:\aris_distributed\nodes.nf
Agent at arisworker1.company.com:14000 added successfully with logical node name n1.
Executing command "add node n2 arisworker2.company.com Clous g3h31m" in line 2 of file c:\aris_distributed\nodes.nf
Agent at arisworker2.company.com:14000 added successfully with logical node name n2.
Executing command "add node n3 arisworker3.company.com Clous g3h31m" in line 3 of file c:\aris_distributed\nodes.nf
Agent at arisworker3.company.com:14000 added successfully with logical node name n3.
Validation of the nodes n1, n2 and n3 completed without errors.
ACC+ >
```

From here on, we will assume that you added ACC to your PATH and are working on a shell whose current directory is your project directory.

You can now run the "list nodes" command once more. All nodes should be listed and show an "OK" state as follows:

```
ACC+ >list nodes
3 nodes:
n1 : arisworker1.company.com (14000) OK
n2 : arisworker2.company.com (14000) OK
n3 : arisworker3.company.com (14000) OK

ACC+ >
```

⚠ **Important:** From now on, whenever we say "start ACC", always pass the node file with the -n switch in addition to any other command line switches mentioned.

3.10   Preparing and using a remote repository

When installing ARIS on a single node with the setup, the setup program will start a repository server, i.e., a simple web server that allows the agent to download the runnables as they are found on the ARIS DVD image.

When installing a distributed environment with ACC, there is no setup program that will do this, so we need to provide a remote repository from which the agents can download the runnables to their local repositories.

In many customer environments, you will have an FTP or web server available, which can be used as a remote repository. Assuming that such a server is available to you, first you need to create a subdirectory in that server's document root (i.e., the directory from which the server serves files) that will hold the runnables (e.g., "arisrepo").

Next, copy the contents of the Setup_Data directory, which can be found on the installation DVD at <DVD_Root>\Setups\Windows\Setup_Data (which usually contains only a folder named "com", as you can see in the screenshot below), to the directory you just created:

FIGURE 17
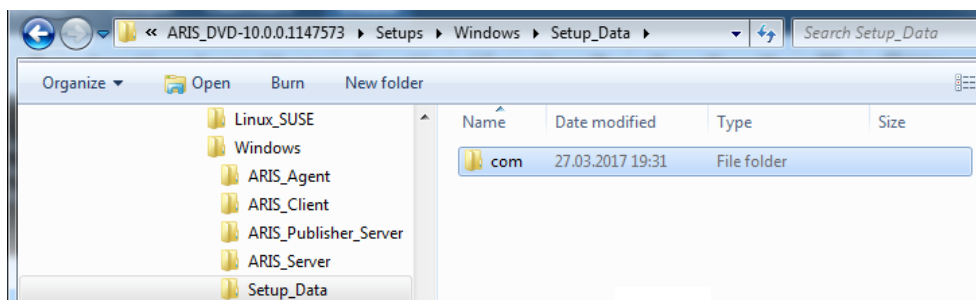
For example, assuming that you plan to use an HTTP server that can be reached via the URL "http://internal.company.com" and whose document root is the directory c:\www, and you created the "arisrepo" folder inside this location, you would copy the content of the Setup_Data folder to this folder:

FIGURE 18
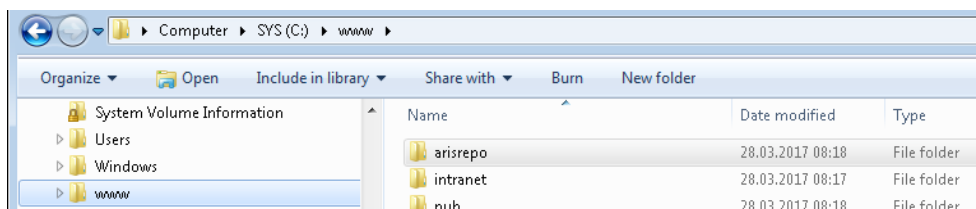
So that afterwards the content of the folder c:\www\arisrepo looks like this:

FIGURE 19
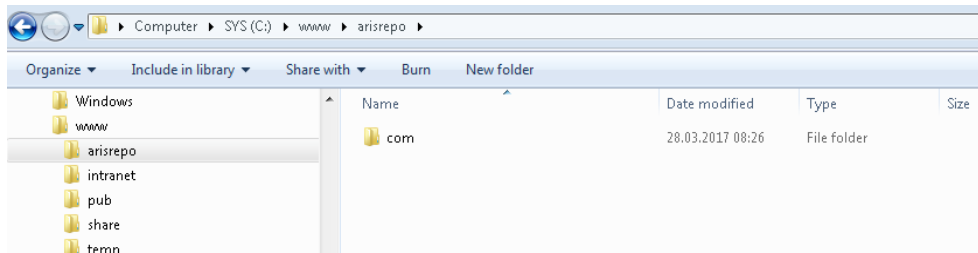
If you now access this directory through a browser (in our example, the URL would be "http://internal.company.com/arisrepo") and assuming that directory browsing is enabled in your web server, you should now see a page similar to this:

FIGURE 20

> ### ⓘ Installing your own remote repository server
>
> If you do not have an FTP or web server available to use as repository server, you can install and run your own server. The simplest approach is to install the Apache HTTPD web server. On Linux machines, you can usually install it using the package manager of your distribution. On Windows, you can download a binary build of Apache from various places on the web. The following is a quick guide about how to install Apache HTTPD on a Windows machine. It was done using a binary build from apachelounge.com (Apache 2.4.25 win64 VC11), downloaded from https://www.apachelounge.com/download/VC11/ (Jan 28th, 2017).
>
> > ⚠ Note that the suggested source for Apache binaries is an external page not related to Software AG in any way. As always, be careful when downloading and running executables from 3rd party pages (i.e., run a virus scanner on the files first).
>
> > ⚠ The Apache server setup with the following steps has not been secured against access in any way, so do not run it on a machine exposed to the public internet.
>
> First unzip the ZIP file with the Apache binary distribution to a directory (for example under "Program Files", "c:\Program Files\Apache\httpd", we will refer to this directory as <apacheDir>).
> Go to <apacheDir>\apache24\conf and open the file httpd.conf in an editor.

Search for the following line

```
ServerRoot "c:/Apache24"
```

and change it so that it points to the "Apache24" subdirectory inside the directory to which you unzipped the ZIP file:

```
ServerRoot "C:\Program Files\Apache\httpd\Apache24"
```

Next, choose a directory that will be your "document root", i.e., the directory where the files served by Apache will be located. As in the example above, we will use "c:\www". First create this directory if it does not exist, and also create the "arisrepo" subdirectory.
In httpd.conf, search for the following line

```
DocumentRoot "c:/Apache24/htdocs"
```

and change it so that it points to your new document root directory, i.e.

```
DocumentRoot "c:/www"
```

(Note that you can use forward and backward slashes in the Apache config files on Windows as path separators).
Then add a "Directory" section for the directory to which you copied the contents of the Setup_Data directory (in our example this was "c:\www\arisrepo), as follows:

```
<Directory "c:/www/arisrepo">
   AllowOverride None
   Require all granted
   Options +Indexes
</Directory>
```

This will give unrestricted access to the contents of that directory and also enable directory listings.That is, when accessing a page that corresponds to a directory in the document root and there is no index file, Apache will generate a clickable index, as it was shown in the screenshot above.
Do not forget to save the httpd.conf file.

You can now start Apache via the httpd.exe file located in <apacheDir>\apache24\bin. If everything works fine, you should see a Windows cmd popping up (and staying open).
If there is a problem (usually you have a typo in the config file or the Apache port - by default port 80 - is already occupied on your machine), the window will show briefly and close immediately, giving you no time to see an error message. In this case, open a command prompt in the <apacheDir>\apache24\bin directory and start Apache by typing httpd.exe in the prompt. You will then be able to see the error message.

After Apache is started, you will be able to access it in any browser using the following URL:

http://<hostnameOfMachineRunningApache>:<apachePort>/<arisRepositoryDirectory>

Assuming that the machine on which Apache is running is named "internal.company.com"

and you did not change the port from the default 80, and you named the directory under your document root to which you copied the content of the Setup_Data directory "arisrepo", the URL will be:

http://internal.company.com/arisrepo

(Note that you can omit the port if you use the default port 80 of the http protocol).

This is also the URL that you will specify as repository URL in the next step.

3.11 Making the agents on the worker nodes use the remote repository

To tell the agents on your worker nodes to use this server as the remote repository, start the ACC and tell all agents about the repo server typing the command:

```
on all nodes set remote.repository.url="<URL_of_Repo_Server>"
```

For example

```
on all nodes set
remote.repository.url="http://internal.company.com/arisrepo"
```

3.12   Get and use a generated.apptypes.cfg file with your ACC

To do anything useful with our ACC, we need to provide it with another file (in addition to our node file) - the generated.apptypes.cfg file. We already saw this file when we looked at the "common" ACC command line switches as they are added to the start menu shortcut that is created by the single-server setup. This file contains the configuration of ACC, in particular it contains information that makes it "know" the technical names and versions of the runnables to install. The path to this file can be specified when starting ACC with the "-c" command line switch.

First we need to get this file fitting to the version of ARIS you want to install. You can find this file on the ARIS DVD image at <DVD_Root>\Setups\Windows\ARIS_Server.

Copy this file to your project directory (i.e., right next to your node file).

If you have not done so already, exit the ACC console and start it again, this time passing both the node file (with the "-n" command line switch) and the generated.apptypes.cfg file (with the "-c" command line switch), as follows:

```
acc -n <pathToNodeFile> -c <pathToGeneratedAppTypesFile>
```

For example, if you opened your command prompt in your project directory (e.g., c:\aris_distributed) and both the node file and the generated.apptypes.cfg file are located there as we suggested, you can run the following command to start ACC with both files:

```
acc -n nodes.nf -c generated.apptypes.cfg
```

If everything works OK, you should see the familiar output we already had above:

```
C:\aris_distributed>acc -n nodes.nf -c generated.apptypes.cfg
Executing command "add node n1 win10vm1 Clous g3h31m" in line 1 of file nodes.nf
Agent at win10vm1:14000 added successfully with logical node name n1.
Executing command "add node n2 win10vm2 Clous g3h31m" in line 2 of file nodes.nf
Agent at win10vm2:14000 added successfully with logical node name n2.
Executing command "add node n3 win10vm3 Clous g3h31m" in line 3 of file nodes.nf
Agent at win10vm3:14000 added successfully with logical node name n3.
Validation of the nodes n1, n2 and n3 completed without errors.
ACC+ >_
```

⚠ **Important:** From now on, whenever we say "start ACC", always pass **both** the **generated.apptypes.cfg file** with the "-c" command line switch and the node file with the "-n" command line switch.

3.13   Securing access to your agents

Unless you are in a private network environment, everyone knowing the default agent username and password can use his or her own ACC to take control of the freshly installed ARIS agents.

To secure your agents, it is recommended that you change the password (and optionally also the username) of each agent to something more secure.

You can do this for all agents currently registered with ACC simultaneously by using the "on all nodes" prefix for the "set" command, as follows:

```
on all nodes set username = <newAgentUserName>
on all nodes set password = <newAgentPassword>
```

 If you have different usernames and passwords on each agent, you must use the "on <nodename>" prefix to change the username and password of each node individually:

```
on <nodename> set username = <newAgentUserName>
on <nodename> set password = <newAgentPassword>
```

For example, to change the username to "admin" and the password to "sup3rT0PS3cr37", you run the commands

```
on all nodes set username = "admin"
on all nodes set password = "sup3rT0PS3cr37"
```

While in ARIS 9 it was necessary to restart the agent for the change to take effect, in ARIS 10 this is no longer needed, the change becomes effective immediately. After you change the username and/or password, your current ACC session will also automatically know about the new username and password and will be able to talk to the agents. However, you must also update your node file accordingly so that next time you start ACC, it is directly given the right credentials. In our example, you will change your node file to

```
add node n1 arisworker1.company.com "admin" "sup3rT0PS3cr37"
add node n2 arisworker2.company.com "admin" "sup3rT0PS3cr37"
```

⚠ The new passwords are now contained in plaintext in your node file - it is therefore important that you restrict access to this file using OS-level file system permissions.

3.14 Our distributed example scenario after performing the preparatory steps

After successfully installing the agents on the three worker nodes used for our example scenario using the installer appropriate for the respective OS, setting up the remote repository and configuring ACC to control all three agents, the current state of our distributed example installation looks like this:

FIGURE 21
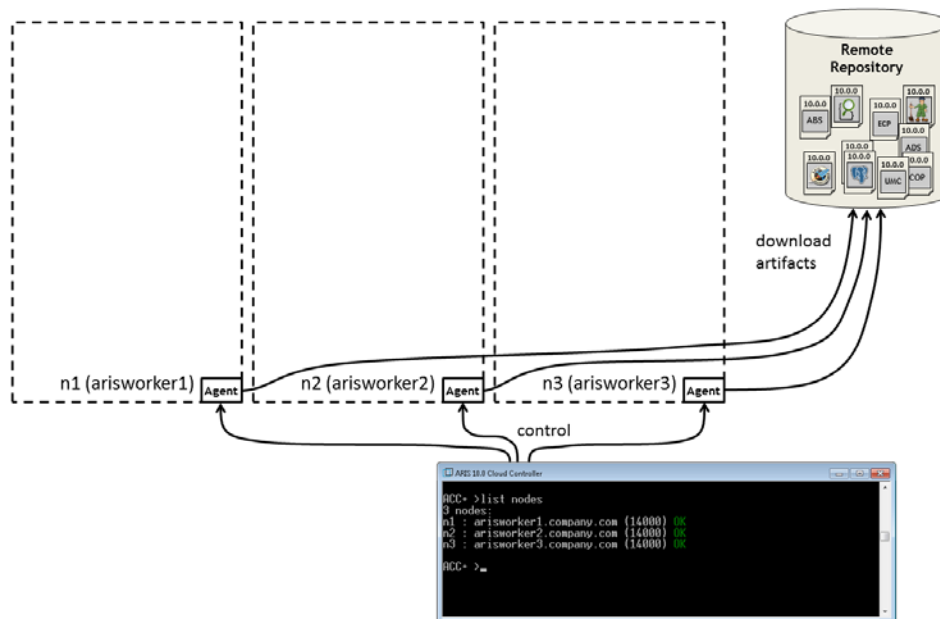
3.15   Overview of the remaining steps

Now that we have successfully prepared our agents, our remote repository, and our working environment, we can finally get started with the actual installation procedure. The next few chapters will explain each major component in the ARIS architecture in more detail, in particular its function within ARIS and how to install it in a distributed environment.

Before you read the next chapter, here is a brief overview of all the remaining steps:

1) You must obtain or create a template file (i.e. a file containing the "configure"and similar commands that will put the runnables on your worker nodes in a reasonable configuration). There are two ways of doing this:
    a) You can obtain an existing template that fits your requirements and the number and hardware sizing of your worker nodes.
    b) You can learn about how ARIS works and then use that knowledge to create your own template, where you decide on which worker nodes you want to put which runnables, if and how you make some or all runnables highly available, etc. This step requires some in-depth knowledge about the ARIS architecture and is covered in detail in the remainder of this guide.

2) You can then start ACC and pass the template file by using the "-f" switch (in addition to generated.apptypes.cfg and nodefile!)
    If everything was done correctly, the "configure" commands should now be executed one after the other.

3) After all configure commands have completed successfully, start ACC (this time passing only the nodefile and the generated.apptypes.cfg file) and type "on all nodes startall".

4) After everything has started, you can access ARIS via any of the worker nodes on which an ARIS **loadbalancer** runnable was configured. If your workers are Windows machines, the ARIS load balancer (ARIS LB) will use the default ports for HTTP (port 80) and HTTPS (port 443) (unless the template you used chose different ports).
    For example, if you have three worker nodes whose hostnames are arisworker1.company.com, arisworker2.company.com, and arisworker3.company.com, and you have a **loadbalancer** runnable configured and running on the first and third machine, you can access ARIS in a browser using the URL http://arisworker1.company.com or http://arisworker3.company.com.

If you decide to follow through this document and learn how to create a template file, you should now prepare a new empty text file in your project directory and name it accordingly (e.g., aris_3_nodes.tp).

All the commands we use in the next chapters to set up the distributed example environment can be run interactively, i.e., by directly typing them or copying and pasting them into the ACC shell as we go. As an alternative, you can add these commands to your template file. This allows you to later reuse the template in other environments as a kind of "canned configuration".

4 Zookeeper clustering and the Zookeeper manager

4.1 Motivation - The role of Zookeeper in the ARIS architecture

This chapter explains how to properly configure the Zookeeper runnable in a distributed scenario.

In the ARIS 9/10 architecture, Zookeeper has several essential roles:

Service Discovery - Any ARIS component (the "runnables") registers itself in Zookeeper with essential information that allows other components to find and access it. The detailed information depends on the type of component, but the minimum information is the hostname (or IP address) and port on which the component can be accessed. Any services used by ARIS that are not provided by a runnable (so-called "external services") also have to be registered in Zookeeper using the ACC service registration functionality (the "register external service", "update external service", "list (external) services", and "deregister external service" commands).

Coordination - Aside from the service discovery functionality, Zookeeper is used to solve a number of common problems in distributed software environments, in particular, but not limited to, the following:

- **Leader election**: Often, certain operations (e.g., background jobs, schema creation, etc.) should be done only once in the system (or once for each tenant). If you have multiple instances of an application in your installation (i.e., "horizontal scale out"), some way is needed for the instances to decide which of them is responsible for such an operation, i.e., which one becomes the so-called "leader". Further, it has to be ensured that if the leader instance fails, a new leader is elected among the remaining instances (i.e., to make sure that the leader "seat" is never vacant for long). For this, the leader election[7] pattern is used. Refer to the Zookeeper manual[8] for details on how leader election can be handled with Zookeeper.
- **Distributed queue**: For some coordination tasks, messages need to be sent reliably between instances. One option is to use a dedicated, persistent, transactional message queue in our infrastructure. However, to keep the footprint small we do not have such a component as a mandatory element of the ARIS infrastructure so far. For basic coordination tasks of the ARIS components with a rather low message volume (i.e., not for messages that transport any user generated events or content), a distributed queue can be implemented on top of Zookeeper[9].

---

[7] Wikipedia: Leader Election https://en.wikipedia.org/wiki/Leader_election

[8] Zookeeper Recipes and Solutions: Leader
Election https://zookeeper.apache.org/doc/trunk/recipes.html#sc_leaderElection

[9] Zookeeper Recipes and Solutions:
Queues https://zookeeper.apache.org/doc/trunk/recipes.html#sc_recipes_Queues

- **Locking**: Some (usually long-running) operations need the guarantee that no other operation interferes with them. For example, a UMC LDAP import should be running only once in the system for each tenant at any given time. To prevent such operations from occurring concurrently, locking is used, which can also be handled through Zookeeper[10].

Due to these essential functionalities handled through Zookeeper, if an ARIS installation is to be made highly available, the Zookeeper service itself has to be made highly available, because any outage of this service will lead to the system becoming unresponsive and/or crashing (almost) immediately.

4.2 Zookeeper ensembles & failover

To make Zookeeper highly available, several Zookeeper instances can be clustered to form a so-called ensemble. In an ensemble, **all member instances will hold the full set of data** that is in the repository and all members of a Zookeeper ensemble keep the content of their repositories in sync with each other. For this to work, each Zookeeper instance needs to know all other ensemble members, as indicated in figure 13 below.

FIGURE 22: A THREE-NODE ZOOKEEPER ENSEMBLE
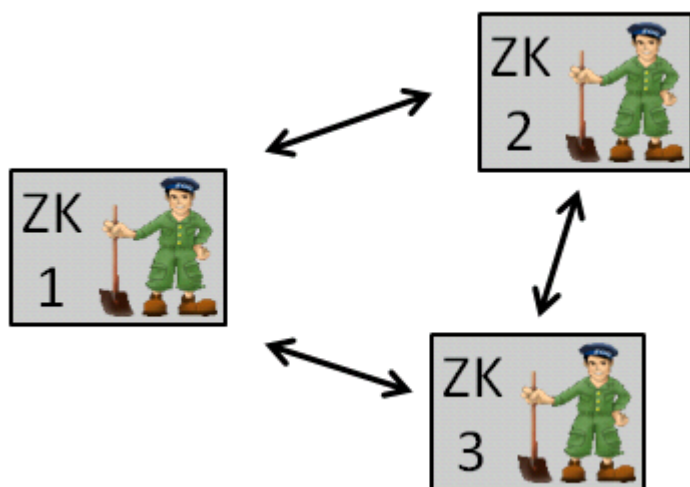
Details on how this knowledge is established are given in the next section.

So whenever a write operation occurs, the ensemble members communicate with each other to ensure that all agree on the outcome of the operation. The Zookeeper master instance is responsible for the coordination of this process. At any given time there must be exactly one instance appointed as master. So when you compare this to our discussion of replica consistency in chapter 2, Zookeeper works with an entire instance being named the master, instead of having masters for individual data items. While this means that all writes have to be coordinated by this instance, this is not a problem for the usual scenarios Zookeeper is used for. The master is chosen in a voting process, which can only occur if a true majority of

---

[10] Zookeeper Recipes and Solutions: Locks https://zookeeper.apache.org/doc/trunk/recipes.html#sc_recipes_Locks

the Zookeeper instances is available and can achieve a so-called quorum (>50%). Without a quorum, there will be no master, and without a master, no Zookeeper operations can happen. In other words, any Zookeeper operation (in particular writes, but also reads) can only happen if more than half (>50%, not ≥50%!) of the ensemble members are working OK and can communicate with each other and thus can "cast their votes" in the master election process. This means that a Zookeeper ensemble of size n can tolerate roundDown((n-1)/2) instances failing before the entire ensemble becomes unavailable, as indicated by the following table:

Total instances	1	2	3	4	5	6	7	8	9
Tolerated failed instances	0	0	1	1	2	2	3	3	4

Note that in particular adding *a second instance* to your ensemble *will not improve the failure tolerance of your Zookeeper ensemble* (on the contrary - the likelihood that one instance out of two fails is higher than that of one out of one failing.). Furthermore, note that having an even number of instances will not increase your level of failure tolerance compared with having an ensemble with one instance fewer.

 However, having as many Zookeeper instances as possible is not an appropriate solution.The reason is that the communication overhead for the master election procedure and in particular for the synchronization of the repository content during write operations increases with each additional instance (as it has to be made sure that all active instances have the same content in their repo). In general, for all but the most ambitious ARIS deployments, a Zookeeper ensemble consisting of three instances is perfectly fine. As a general rule of thumb the Zookeeper documentation states that when you put more than seven instances into an ensemble, the performance of the ensemble will actually decrease compared to a smaller one.

> ℹ **Zookeeper observer instances**
>
> Zookeeper also enables you to add so-called observer instances that do not participate in the master election process, but can otherwise be used by clients just as regular instances. This allows you to have a larger number of Zookeeper instances without increasing coordination overhead in particular to give clients that are placed remotely from the main data center (i.e., where the voting ensemble members are located) a local instance with which to communicate. While using observer instances can be beneficial in certain deployment scenarios, for ARIS deployments we never found a good reason to use them.

4.3   Failover scenarios

Let us now have a look at a few failure scenarios. Assume that in our three-node ensemble from the figure above, instance 2 suddenly fails (e.g., the Zookeeper runnable crashes, the entire machine goes down, the data center it is located in goes up in flames, etc.), as indicated in figure 23 below:

FIGURE 23: OUTAGE OF ONE INSTANCE IN A THREE-NODE ZOOKEEPER ENSEMBLE
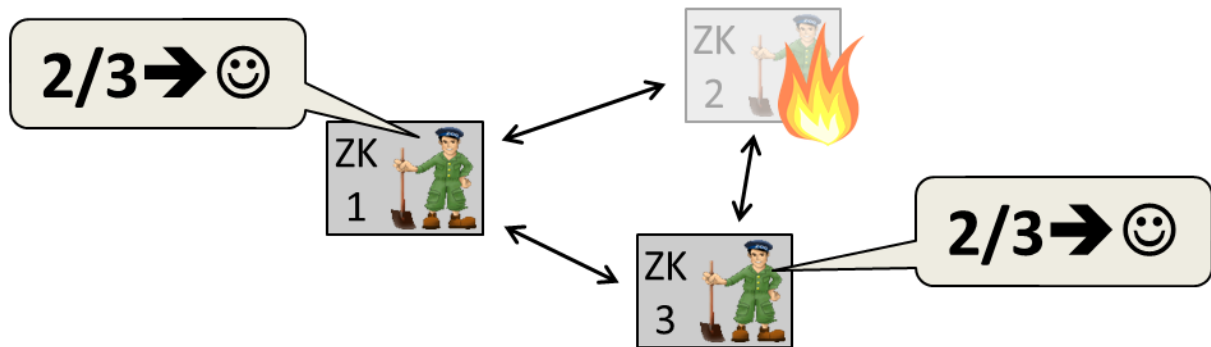
Note that the remaining instances 1 and 3 together still have 2/3 of the votes They can therefore successfully elect a master instance among them and continue working normally (i.e. serve read *and* write requests). As soon as instance 2 becomes available again, it will rejoin the ensemble and sync its content to that of instances 1 and 3.

Let us now consider the apparently similar scenario shown in figure 24 below.

FIGURE 24: NETWORK OUTAGE (PARTITIONING) IN A THREE-NODE ZOOKEEPER ENSEMBLE
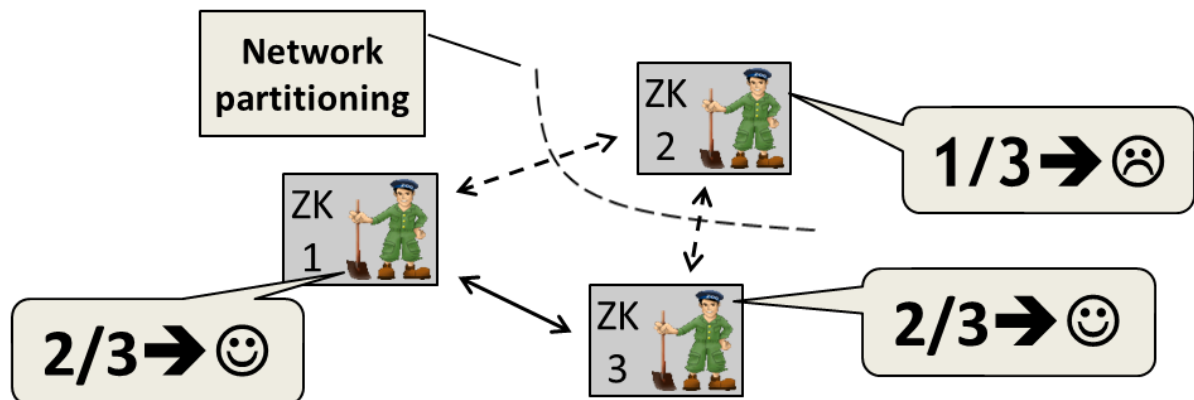
Here, instance 2 continues working fine, however, due to a network partitioning, it can no longer communicate with the other instances. The basic behavior is the same as before: the larger partition with instances 1 and 3 still has a quorum, so the instances can elect a master among them and can continue serving requests. Instance 2 in the smaller partition, while fully functional, cannot collect enough votes to elect a master and does not serve any requests. That way, the dreaded split-brain problem is avoided: assume for a moment that

Zookeeper did not require a quorum to allow read and write operations. In that case, the smaller partition (consisting of only instance 2) could still continue working and could serve clients that can still reach it (because they are on the same side of the network partition). That way, the content of the repositories of the instances in the two partitions could be modified independently and thus diverge. While this would not be noticed as long as the network partitioning persists, if the network partitions were later resolved, it would not be clear which instances hold the "true" state of the repository (instances 1 and 3, or instance 2) and the repositories could not be consolidated into one consistent state, a situation that has to be avoided at all cost, because it could not be resolved automatically.

4.4   Setting up a Zookeeper cluster (or "ensemble") manually

To make each ensemble member know all other instances, each instance must be given a list of the hostnames or IP address of *all* ensemble participants and their respective server ports (named server port A and server port B), using the indexed, multi-valued "server" configuration parameter. The values of this parameter have the format

```
<hostname>:<serverPortA>:<serverPortB>
```

Being "indexed" means that instead of supplying multiple values for that parameter simply as a comma-separated list (e.g., x= a,b,c), you specify the *individual values* as *individual configure parameters* by appending ".<index>" to the parameter key (e.g.: x.1 = a x.2 = b x.3 = c).

For example, in a three-node ensemble, where Zookeeper instances are installed on the nodes arisworker1.company.com, arisworker2.company.com, and arisworker3.company.com and are using the standard ports for server port A and server port B (for ARIS 10 these are the ports 14285 and 14290, respectively), you must set the configure parameters for each instance:

```
server.1=arisworker1.company.com:14285:14290
server.2=arisworker2.company.com:14285:14290
server.3=arisworker3.company.com:14285:14290
```

⚠️ In previous versions (<10), the server parameter could be specified using a single multi-valued parameter, for example:
```
server="arisworker1.company.com:14285:14290", "arisworker2.company.com:14285:14290",
"arisworker3.company.com:14285:14290"
```

While multi-valued parameters still exist in ARIS 10, for technical reasons they cannot be used anymore for setting the server parameter(s) of a Zookeeper ensemble.

In addition to providing the proper values of the server.<index> parameters, each instance must be told which member in that list the instance itself is, using the "myid" parameter. For example, for the instance located on node aris1, you set myid to 1. Note that this way you also implicitly tell each instance which server ports it should open.

4.4.1 Example

Assume you want to have a Zookeeper cluster including three nodes with the hostnames arisworker1.company.com, arisworker2.company.com, and arisworker3.company.com, and which you registered in ACC as n1, n2, and n3, respectively.

You configure your three Zookeeper instances as follows (note the use of a "\" as the last character in a line to make the command continue on the next line).

```
on n1 configure zoo_m zoo \
  myid=1 \
  server.1=arisworker1.company.com:14285:14290 \
  server.2=arisworker2.company.com:14285:14290 \
  server.3=arisworker3.company.com:14285:14290


on n2 configure zoo_m zoo \
  myid=2 \
  server.1=arisworker1.company.com:14285:14290 \
  server.2=arisworker2.company.com:14285:14290 \
  server.3=arisworker3.company.com:14285:14290


on n3 configure zoo_m zoo \
  myid=3 \
  server.1=arisworker1.company.com:14285:14290 \
  server.2=arisworker2.company.com:14285:14290 \
  server.3=arisworker3.company.com:14285:14290
```

Observe that the values for the three server.<index> configure parameters are the same across all instances. Also observe that only the value of the myid parameter differs, telling each instance which "position" among the list of servers it holds.

The obvious disadvantage of this manual configuration is that a) you can get it wrong and b) you cannot simply use the same provisioning template again in a different environment, because the server.<index> configure parameters now contain values (the hostnames or IP addresses of the machines) that depend on the actual deployment environment.

4.5   Setting up a Zookeeper cluster with ACC's built-in Zookeeper manager

To make setting up a Zookeeper ensemble easier and keep templates portable, ACC contains the "Zookeeper manager" component, which offers a set of high-level commands to simplify initial setup and later changes to an ensemble.

4.5.1   Configuration

The only ACC configuration parameter required by the Zookeeper manager is "zkmgr.zookeeper.app.type", which must be set to the name of the app type that will be used for configuring Zookeeper instances. Usually, this will be the app type "zoo_m".

Note that if an app type with the name given for this parameter is not registered in ACC, the Zookeeper manager will not work.

4.5.2   Commands

The following section describes all ACC commands that relate to ACC's Zookeeper manager component. Note that in general the command syntax allows you to use the keywords "zookeeper" or "zk" interchangingly.

4.5.2.1   Add zookeeper instance

The "add zookeeper instance" command stages the addition of a new Zookeeper instance to the installation's Zookeeper ensemble on the specified node (or to the current node, if no node name is specified). It allows you to optionally specify the instanceId to be used for the new Zookeeper instance. If none is specified, the cloud controller will automatically choose an instanceId. Note that the change is not submitted directly when executing the command, but it is only "staged". To apply the change, you must use the "commit zookeeper changes" command.

Syntax:

<nodeSpec>? "add" ("zk"| "zookeeper") ("master"| "observer")? "instance"? ("using" "instanceId"? <instanceId>)?

Examples:

```
add zk
```

This stages the addition of a new Zookeeper instance on the current node.

```
on n1 add zookeeper
```

This stages the addition of a new Zookeeper instance on node n1.

4.5.2.2   Remove zookeeper instance

The "remove zookeeper instance" command stages the removal of a Zookeeper instance from the installation's Zookeeper ensemble. The Zookeeper instance to be removed can be identified either by specifying the node name and the instanceId of the Zookeeper instance or by specifying the myId of the Zookeeper instance (i.e., its sequential number among all instances of the ensemble).

Syntax:

<nodeSpec>? "remove" ("zk" | "zookeeper") "instance"? ("instanceId"? <instanceId>)?
| "remove" ("zk" | "zookeeper") "instance"? ("myId"? <integer>)

Examples:

```
remove zk myid 3
```

This stages the removal of the Zookeeper with myid 3 from the ensemble.

```
on n2 remove zookeeper zoo2
```

And this removes the Zookeeper instance with instance id zoo2 on node n2.

4.5.2.3   Commit zookeeper changes

The "commit zookeeper changes" command applies all currently "staged" or "pending" changes to the Zookeeper ensemble.

Syntax:

"commit" "pending"? ("zk" | "zookeeper") ("change" | "changes") "force"?

Example:

```
commit zookeeper changes
```

4.5.2.4   Discard zookeeper changes

The "discard zookeeper changes" command discards all currently pending changes in the Zookeeper ensemble.
Syntax:

("discard" | "reset") "pending"? ("zk" | "zookeeper") ("change" | "changes")

Example:

```
discard pending zk changes
```

4.5.2.5   List zookeeper instances

The "list zookeeper instances" command displays all Zookeeper instances of the ensemble with their configuration.

Syntax:

"list" ("zk" | "zookeeper") "instances"

Example:

```
list zk instances
```

4.5.2.6   Show zookeeper changes

The "show zookeeper changes" command displays all currently pending changes to the Zookeeper ensemble that will be applied when using the "commit zookeeper changes" command.

Syntax:

"show" "pending"? ("zk" | "zookeeper") ("change" | "changes")

Example:

```
show pending zookeeper changes
```

4.5.2.7   validate zookeeper changes

The "validates zookeeper changes" command performs a validation of all currently pending changes to the Zookeeper ensemble, i.e., it checks whether the changes will result in a working ensemble.

Syntax:

"validate" "pending"? ("zk" | "zookeeper") ("change" | "changes")

Example:

```
validate zk changes
```

4.5.2.8   validate zookeeper ensemble

The "validate zookeeper ensemble" command performs a validation of the current state of all already configured Zookeeper instances. In particular, it checks if all instances form a single consistent ensemble. Any currently "staged" or "pending" changes are not considered by this command.

Syntax:

"validate" "pending"? ("zk" | "zookeeper") ("change" | "changes")

Example:

```
validate zookeeper ensemble
```

4.5.3   Usage example

If you want to set up the three-node ensemble from our example above using the Zookeeper manager instead of doing it manually, you execute an "add zookeeper instance" command for each instance, specifying the node on which it should be placed. Remember that this will not yet make any changes to the ensemble, but only "stages" the intended changes. To apply the changes, you execute the "commit zookeeper changes" command. So, assuming that your three worker nodes were registered as n1, n2, and n3, the following four commands will automatically set up a working Zookeeper ensemble and start all three instances.

```
on n1 add zk
on n2 add zk
on n3 add zk
commit zk changes
```

4.6   Bootstrapping the Zookeeper connection

In this section, we have learned so far that Zookeeper is an essential element of the ARIS architecture and that it is - among other things - used to allow runnables to find each other. The next section explains how runnables find Zookeeper.

4.6.1   How do runnables find Zookeeper?

To keep things simple for single-node installations, by default, a runnable will try to access Zookeeper on the local machine on the default port of the Zookeeper runnable (which is 2181 for ARIS 9.x and 14281 for ARIS 10.x).

This default mechanism will also be enough for small distributed scenarios, where each node will hold an instance of the Zookeeper ensemble. Because the first Zookeeper instance on a node will be listening for client requests on the default port, all runnables on that node will find their local instance. However, if we leave things like that, there is already a bit of a restriction here: if a Zookeeper instance fails, all runnables on the same node will also lose their Zookeeper connection (even if the other Zookeeper instances on other nodes still have a quorum and work fine) and will eventually cease to work - in other words, the local Zookeeper instance becomes a single point of failure for all runnables on that node. Figure 25 below illustrates this situation. Here we have a simple distributed scenario consisting of three nodes, each node holding a single Zookeeper instance of a three-node ensemble plus a selection of other runnables.

FIGURE 25
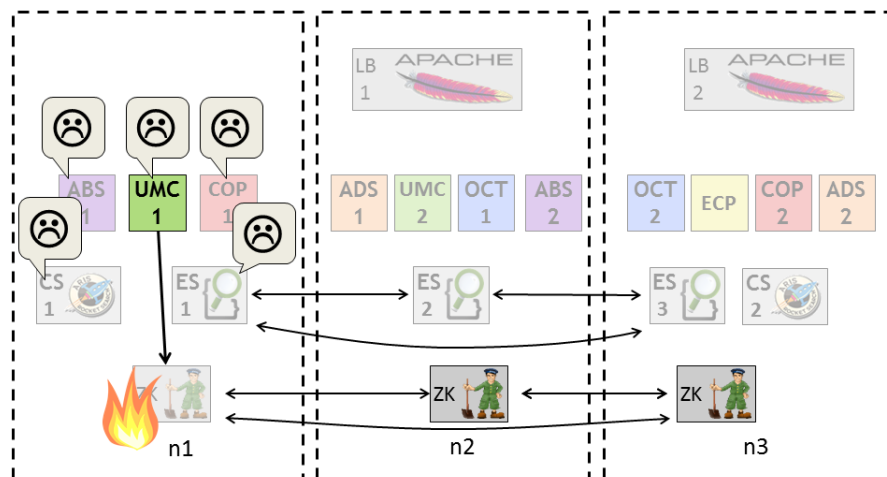
In the distributed scenario sketched here, the Zookeeper instance on node n1 fails, leaving all runnables on the same node without a Zookeeper connection, despite the ensemble as a whole still being available, as the Zookeeper instances on nodes n2 and n3 still have a quorum.

For small distributed scenarios without the highest requirements regarding high availability, this is in practice not a strong limitation, because Zookeeper is usually quite stable and even if it does crash, the agent will restart it within a few seconds. Most of our applications

tolerate a short Zookeeper outage quite well (they will usually just stop responding until it is back, i.e., a request to the app will "hang" for a moment). Further, when aiming for a highly available ARIS installation, you must tolerate failure of an entire node (or loss of network connectivity, etc.).

However, if this problem is not addressed, the installation as a whole will be less robust than it could be.

It is obviously preferable to make each runnable know about all Zookeeper instances, so that they can switch to another one if their local instance fails, and, of course, for more complex deployments where we do not have a Zookeeper instance on each node. Consider for example the scenario sketched below:

FIGURE 26: A NODE WITHOUT A LOCAL ZOOKEEPER INSTANCE
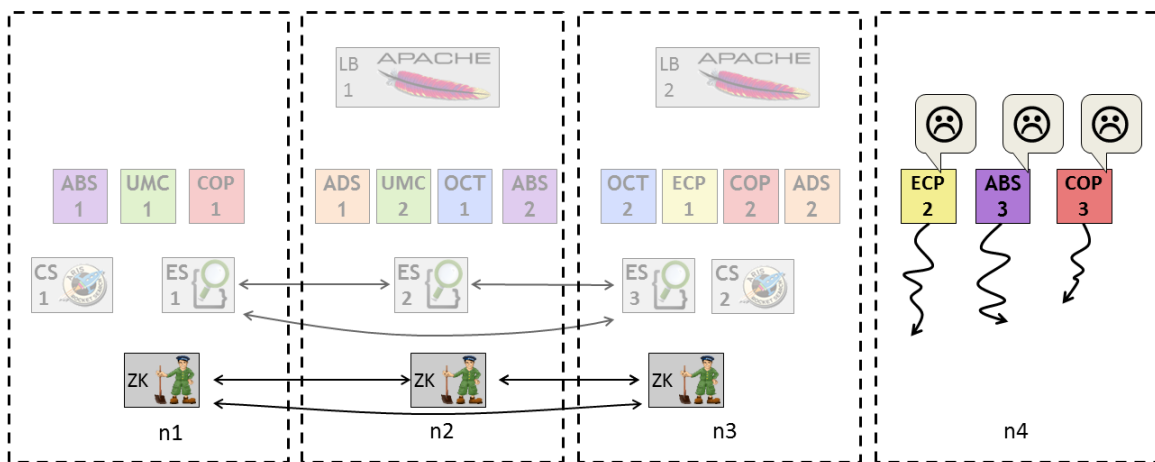
Here, we added a fourth node to our previous example scenario to have additional capacity for scaling out some of our runnables, so that we can handle a higher request load. However, since we learned that having an even number of Zookeeper instances is not useful, we decided to not add a Zookeeper instance to this node as well. By default, however, this will leave our new runnables without a Zookeeper connection, and they will not be able to work and handle user requests.

4.6.2   The zookeeper.connect.string configure parameter

To resolve this problem, each runnable can be told where to find Zookeeper instances with the zookeeper.connect.string parameter. The zookeeper.connect.string contains a comma-separated list of the hostnames and client ports of Zookeeper instances. Assuming that our four nodes n1 to n4 above have the hostnames aris1 to aris4, and assuming that our three Zookeeper instances on nodes n1 to n3 are left at the default client port (14281), the proper value of the Zookeeper connect string in this installation will be
aris1:14281,aris2:14281,aris3:14281

We can now set this value with a reconfigure command on our three additional runnables on node n4 as show here:

FIGURE 27: SETTING THE ZOOKEEPER.CONNECT.STRING PARAMETER
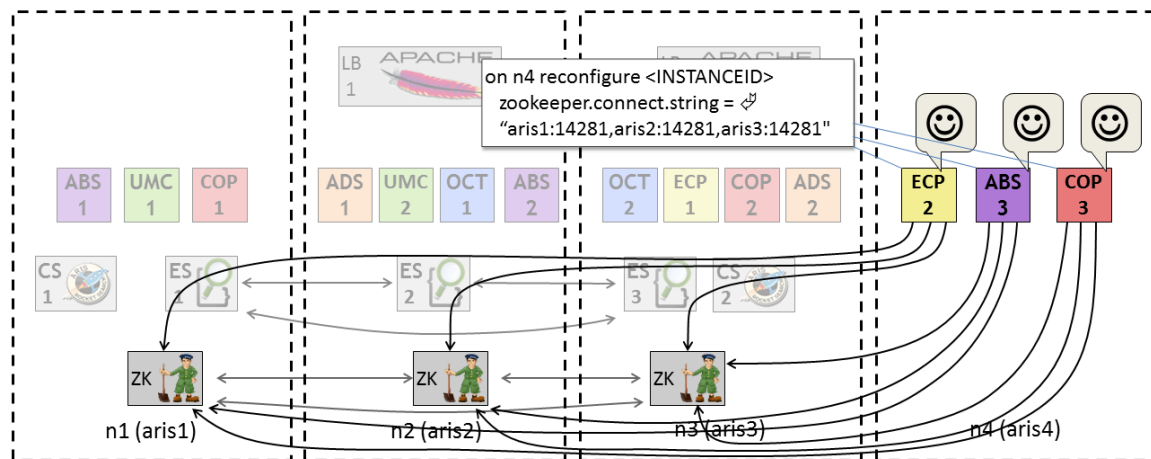
(To improve failure tolerance, i.e., to not have the local Zookeeper instances on nodes n1 to n3 as a single point of failure for all runnables on that node, we must do the same for all runnables on all nodes, as discussed above).

While using the zookeeper.connect.string parameter allows you to remedy the problem of Zookeeper connectivity, this is not very convenient and is also error-prone, because you must maintain this parameter on all runnables. And again, the value of this parameter - while the same for all runnables in an installation - depends on the actual deployment environment, as the zookeeper.connect.string parameter contains the hostnames (and not the logical node names) of the nodes involved and thus will need to be changed for each deployment environment, again making templates not portable.

4.6.3 Automatic setting of zookeeper.connect.string (≥ 9.8.2)

To resolve this problem, with version 9.8.2, a feature was introduced to make this more convenient:

an agent configuration parameter, "zookeeper.connect.string", was added. First, the value of this parameter is used by the agent itself to create its own Zookeeper connection (needed for the agent to be able to perform certain commands). Second, the value of this parameter is passed to the runnables (and the runnables' plugins) on startup, making it no longer necessary to manually set the zookeeper.connect.string configure parameter on the runnables.

> ℹ **Some background information about passing the zookeeper.connect.string to the runnables on startup**
>
> The agent will pass its own zookeeper.connect.string to the runnables when it starts them by setting a Java system property, which works only for Java runnables. All Java-based runnables with ZKC inside will then be able to use this zookeeper.connect.string value to establish their Zookeeper connection, but will only do so, if the zookeeper.connect.string has not been set as a regular config parameter for this runnable (or has been set to the empty string or has been set to the default "localhost:14281").
>
> For non-Java-based runnables (i.e., the Apache-HTTPD-based load balancer, Postgres, and - for ARIS 9.x only - CouchDB (for ARIS 9.x)) and for Java-based third-party runnables that do not have ZKC inside (Elasticsearch), the registration of the runnable in Zookeeper is done by the runnables' associated agent plugin. These plugins are also given the agent's zookeeper.connect.string to open their Zookeeper connection.

Further, under certain conditions, ACC can automatically set the zookeeper.connect.string parameter of all agents that are registered with it, as illustrated in the figure below:

FIGURE 28: AUTOMATIC SETTING OF THE ZOOKEEPER.CONNECT.STRING BY THE ACC
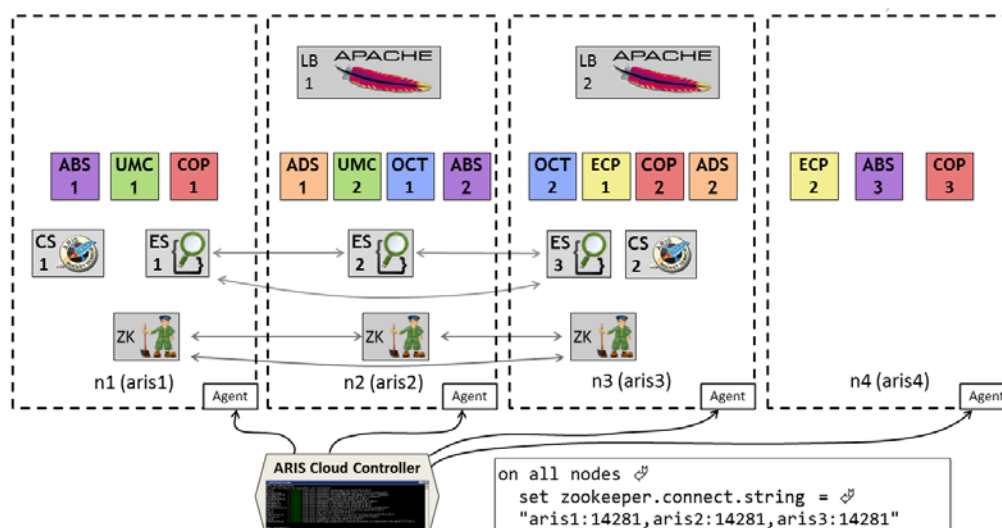
In particular, there must be only one Zookeeper ensemble across all nodes, and this ensemble must be valid and consistent. Further, the nodes on which the Zookeeper instances are running must be registered with a "real" hostname or IP address, not with a loopback address (e.g., "localhost", "127.0.0.1" etc.).

If these conditions hold, ACC will update the zookeeper.connect.string of the agents
a) whenever a new node is added (and the conditions still hold)
b) whenever an ensemble change is performed via the Zookeeper manager
Automatic updating is enabled by default, but can be disabled by setting the ACC config parameter zkmgr.maintain.agent.zookeeper.connect.string to "false" (remember that like all ACC settings, this setting is not persisted and must be re-done when restarting ACC).

4.7   Zookeeper in our example scenario

In our example scenario, where we have three nodes n1 to n3, we can now just use the basic Zookeeper manager "add zk" and "commit zk changes" commands introduced above to add one Zookeeper instance on each node:

```
on n1 add zk

on n2 add zk

on n3 add zk

commit zk changes
```

You can now run these commands directly in your ACC shell and/or you can add them to your template file.

Assuming that everything works OK, you will now have three already started Zookeeper instances that are correctly configured to form a working ensemble that can tolerate the failure of any single instance, as shown in the figure below:

FIGURE 29: THE ZOOKEEPER ENSEMBLE IN OUR EXAMPLE SCENARIO
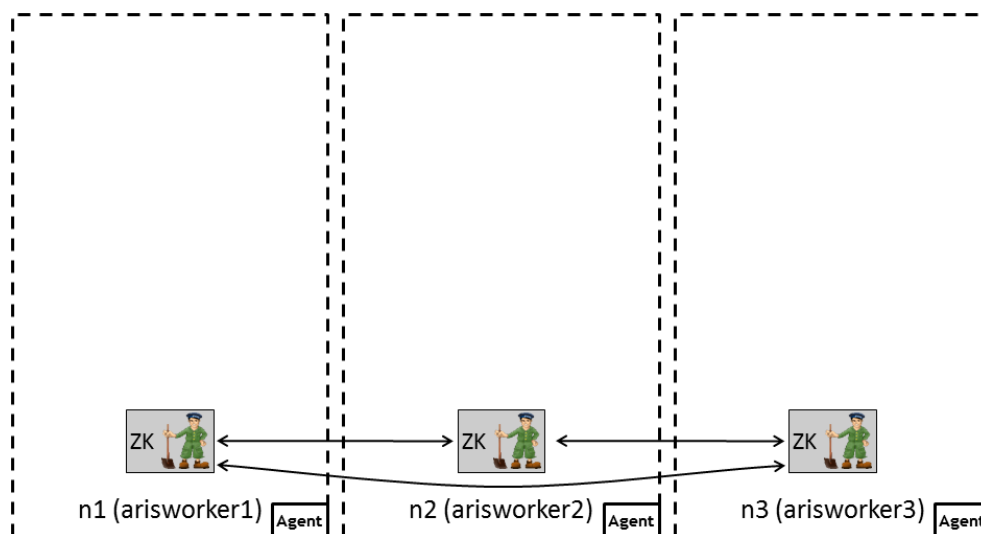
5 Elasticsearch clustering and the Elasticsearch manager

The Elasticsearch manager may only be used in distributed environments to configure a valid Elasticsearch cluster. To avoid major problems, never use it for single node environments. To set up single nodes, use the setup program or a simple executable configuration command.

5.1 Motivation - The role of Elasticsearch

In the ARIS 9/10 architecture, Elasticsearch serves as the backend for several applications: it is used as search index for collaboration and document content. Further, it is the backend used to store all UMC data (users, user groups, licenses, license assignments) and also serves as the backend for all currently valid user sessions. In particular, the essential role it has for UMC makes Elasticsearch a service whose availability is pivotal for the availability of ARIS as a whole. If you want to set up a distributed, highly available ARIS installation, you must set up Elasticsearch in a highly available way.

5.2 Elasticsearch terminology and clustering

Elasticsearch is first and foremost a scalable full-text search engine. It can store arbitrary documents in JSON format, which it can access in real-time, making it effectively also a NoSQL backend.

The basic unit of persistence in Elasticsearch is the **document**, which can be very roughly compared to a tuple in a relational database system. A document consists of several **fields**, which can be roughly compared to a column in a relational database system. However, unlike a column in a relational database, a field can have complex, i.e., nested and repeating elements. A **type** is a category of documents that have similar properties and can be roughly compared to a relation or table in a relational database, however, unlike an RDBMS, documents of one type need not have the same fields. An Elasticsearch **index** is a logical collection of documents (of possibly different types), the closest analogy being a logical database in an RDBMS.

Figure 30 below gives and overview of these basic concepts: Index A consists of multiple documents assigned to two different types, X and Y.

FIGURE 30: ELASTICSEARCH TERMINOLOGY: DOCUMENTS, TYPES, INDEXES, AND SHARDS
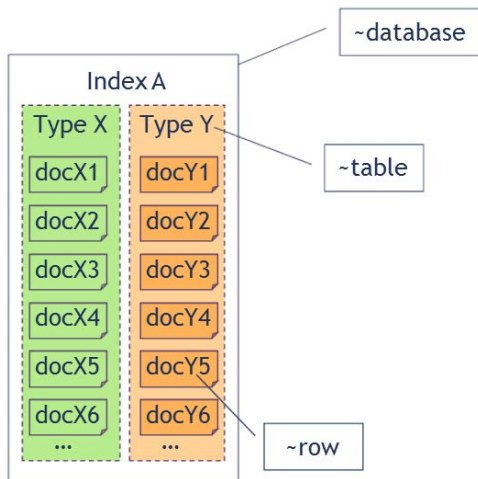
Aside from these logical data structures, Elasticsearch also gives some control over the physical storage of data. In particular, it allows you to split a single index into several physical segments, the so-called **shards**. A shard is the unit of distribution and replication in an Elasticsearch cluster, i.e., the shards of an index can be located on different Elasticsearch instances.

⚠ Note that in official Elasticsearch terminology what we call an "instance" is actually called a "node". We deliberately chose to not use Elasticsearch terminology here to avoid confusion between our meaning of nodes, i.e., as a machine on which an agent is running and which can therefore hold ARIS runnables.

Figure 31 below shows our index from figure 1, and how it can be split into shards.

FIGURE 31: AN ELASTICSEARCH INDEX CONSISTING OF THREE SHARDS
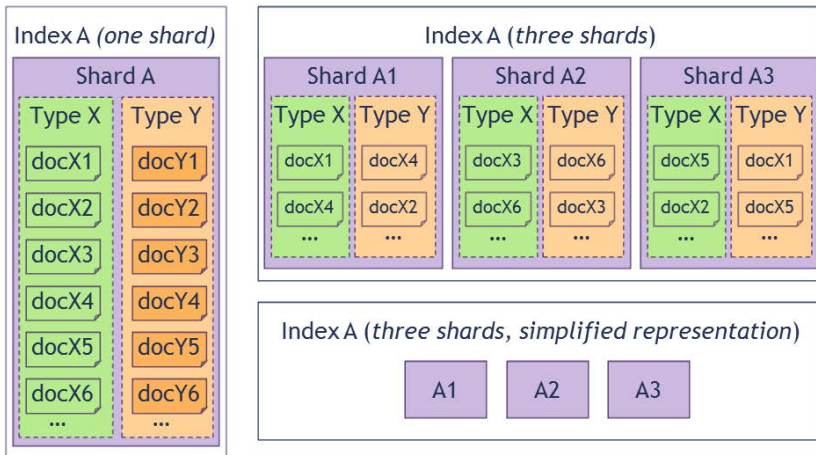
To the left, the index is kept in a single shard. The upper right shows the same index, but this time split into three shards, each containing some of the documents. The lower right finally shows the same index, again with three shards, but in a simplified presentation, where we do not show the types and documents, but only the shards themselves.

The shards of an index can be distributed among the Elasticsearch instances of a cluster, as it is illustrated in figure 32 below.

FIGURE 32: DISTRIBUTION OF THE SHARDS OF AN ELASTICSEARCH INDEX ACROSS A CLUSTER
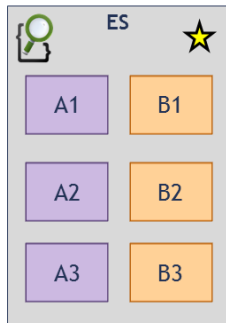
(a) One instance
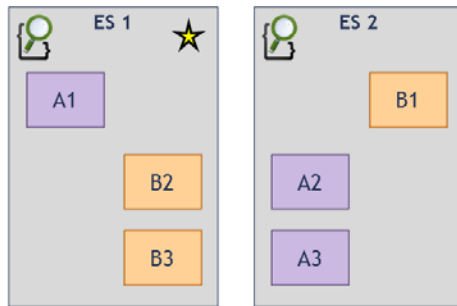
(b) Elasticsearch cluster with two instances
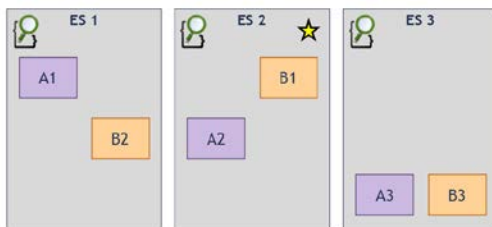
(c) Elasticsearch cluster with three instances

In figure 32 (a), we have a single Elasticsearch instance, with two indexes A and B with three shards each. With only one instance available, all shards of all indexes are held by that instance. Figure 32 (b) shows the same logical indexes, but now in an Elasticsearch cluster consisting of two instances, and a possible distribution of the shards among these instances. Figure 32 (c) finally shows the same two indexes, but this time in a cluster consisting of three instances, and a possible distribution of the shards. Note that Elasticsearch will make sure that the shards are distributed more or less equally across all instances of the cluster. That way, not only the amount of data held by the cluster but also the load of handling requests can be distributed more evenly. Note that clients do not need to know which documents of a cluster reside in which shards - clients can perform requests on any instance and Elasticsearch will then transparently retrieve the data needed to answer the request from the other instances.

Note that in any of the scenarios in figure 32 above, we only have a single copy of each shard (the so-called "primary shard"), so our Elasticsearch cluster is not highly available - when an instance holding a shard becomes unavailable, the index will not be available until the situation is resolved.

One important role we have not yet mentioned is that of the master instance. In each cluster, (even in one consisting of only a single instance) one instance is elected to be the master, which is responsible for allocating the shards to the instances. In our figures, we indicate the master instance with a star symbol.

Just as we discussed in chapter 2 when we looked at the basics of high availability, we need data redundancy. For each shard, Elasticsearch can manage one or more redundant copies of shards, the so-called "replica shards". To control replica consistency, any changes are handled by the respective primary shard that is then responsible for updating the replica shards (by default with **eventual consistency** only). The replica shards can be used to serve read requests and will allow Elasticsearch to recover the data if the instance holding the primary shard of an index fails. Figure 33 shows our two indexes A and B from the example above, but this time configured to hold an additional replica for each shard, and how these replicas can be distributed across the instances of a three-node cluster. In the figure, we indicate the shards that are primary shards by underlining their name and using a bold border.

FIGURE 33: DISTRIBUTION OF PRIMARY AND REPLICA SHARDS IN AN ELASTICSEARCH CLUSTER
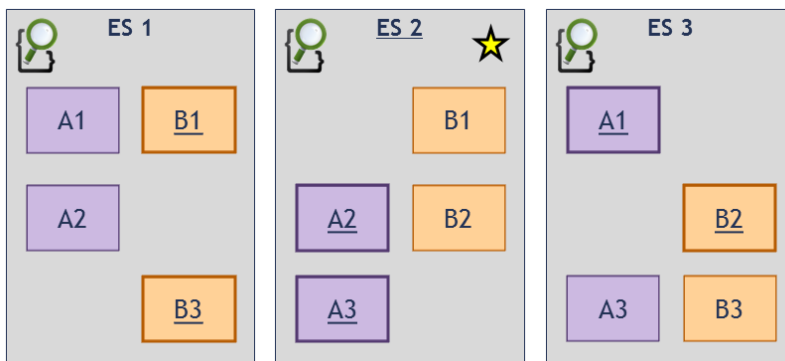
In the state shown in this figure, instance ES 2 has been elected to be the master instance. Observe that we have two copies of each shard, and that these copies are always on different nodes. So if a single instance of our cluster fails, the data is still available.

For example, assume that instance ES 2 (which happens to be the master) fails, as shown in figure 34 below.

FIGURE 34: FAILURE OF ONE INSTANCE IN A THREE-NODE CLUSTER
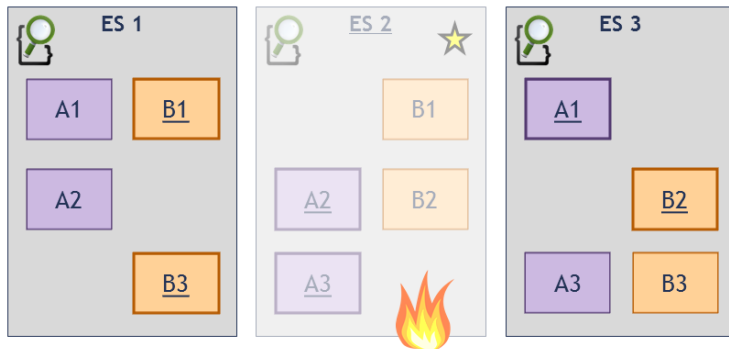
Because the failed instance happens to be the master instance, the very first step is the election of a new master instance. Let us assume that instance ES 3 becomes the new master. Because the failed instance also contained the primary shards of A2 and A3, the new master will first make the former replica shards of the shards that exist in instances ES 1 and ES 3, respectively, the new primary shards. Next it will create new replica shards, which it will allocate on the remaining instances. After a short amount of time, all shards will again have a primary and a replica shard. A possible outcome of the automatic handling of the failed instance is shown in figure 35 below.

FIGURE 35: HANDLING OF A FAILED INSTANCE
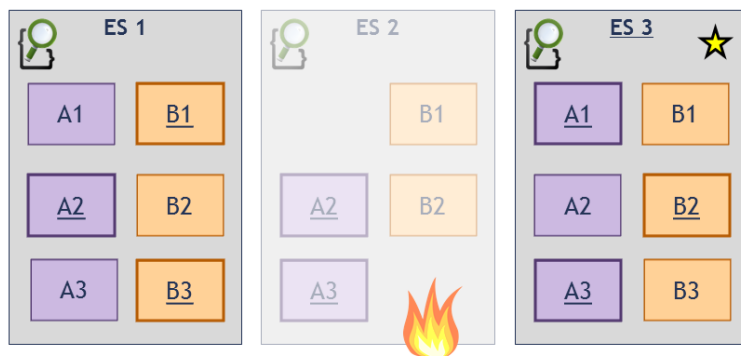
If in the state shown in figure 35 another instance fails, we still have one copy of each shard in the system and thus will not lose any data. However, to avoid the dreaded split-brain problem that we discussed in chapter 2, an Elasticsearch cluster is in general configured to require a quorum of more than half of all instances to elect a new master. With only one out of three instances available, the cluster will therefore become unavailable if, for example, instance ES 1 fails as well.

If in the state shown in figure 35 the failed instance ES 2 becomes available again, it will rejoin the cluster and will be assigned new (primary and replica shards) to again balance the load across all instances.

The key to making Elasticsearch highly available is to configure it properly in order to obtain the behavior that we just sketched. This topic is discussed in the remainder of this chapter.

5.3 Why clustering Elasticsearch became more difficult with ARIS 10

With ARIS 10, the version of Elasticsearch shipped with ARIS has been updated to 2.4, which no longer supports automatic cluster discovery via Zookeeper. Further, the alternative, network-broadcast-based approach (or multicast discovery) is also no longer a standard functionality (and will not work for some environments, in particular Amazon EC2). The only remaining discovery mechanism is unicast discovery, which requires that each Elasticsearch instance that should become part of a cluster is given a list of the hostnames and transport ports of at least a subset (but preferably all) other instances of the cluster, so that it can link up with them. Having to manually maintaining such a list, while technically easily possible with the configure parameter "ELASTICSEARCH.discovery.zen.ping.unicast.hosts", has the significant drawback that a provisioning template must specify installation-specific parameters (the hostnames!) and will no longer be portable. Even if you accept this limitation, setting this parameter can be easily done wrong. In addition, experience with Elasticsearch encountered in customer installations has shown that it is in general difficult to set up a proper Elasticsearch cluster, in particular setting the proper node names, cluster names, the proper quorum, etc.

With ARIS 10 we provide a new functionality as part of ACC, the Elasticsearch manager. Similar to the Zookeeper manager that already comes with ARIS 9.x and that we used in the previous chapter to conveniently set up a Zookeeper ensemble, it considerably simplifies the creation of a working Elasticsearch cluster. But before we discuss the Elasticsearch manager, let us first have a look at how to manually configure an Elasticsearch cluster.

⚠️ **Making Elasticsearch run on Linux**

With ARIS version 10 SR4, ARIS uses Elasticsearch version 5.x instead of 2.x. Elasticsearch 5.x performs a number of initial checks of various system settings (the so-called **bootstrap checks**). If one of these checks fails, Elasticsearch will not start up. These problems can be recognized by checking the runnable's **elasticsearch.log** log file, located in the runnable's log directory (<installDir>/cloudagent/work/work_<ELASTICSEARCH_INSTANCEID>/logs). One setting that is particularly troublesome on Linux systems is the **vm.max_map_count** setting, which in almost all environments will be set too low by default. Elasticsearch requires that this value be set to at least **262144**. The corresponding error message in elasticsearch.log will be:

"max virtual memory areas vm.max_map_count [xyz] is too low, increase to at least [262144]"

(xyz corresponds to the current value set on your node)

You can also check the current value by running the Linux shell command (e.g., via an SSH shell on the node) directly on the node(s) on which the Elasticsearch runnable(s) is/are located:

```
sysctl -n vm.max_map_count
```

You can change it transiently (i.e., only until next reboot) by running the Linux shell command directly on the node(s) on which the Elasticsearch runnable(s) is/are located:

```
sudo sysctl -w vm.max_map_count=262144
```

To change it persistently, as a root user, edit the /etc/**sysctl.conf** file on the node(s) on which the Elasticsearch runnable(s) is/are located and add the following line to the file: `vm.max_map_count=262144.`

5.3.1   Manually setting up an Elasticsearch 2.x cluster

5.3.1.1   Configuring an Elasticsearch instance to find the other cluster members

Elasticsearch's unicast discovery is based on the gossip concept: a node joining the cluster needs access only to at least one running member of the cluster from which it can then retrieve information about the other members.

With the configure parameter ELASTICSEARCH.discovery.zen.ping.unicast.hosts set for an Elasticsearch runnable, you can provide such a list of nodes. The structure of this parameter is a comma-separated list of the hostnames and ports (with the port separated from the hostname by a colon) of the other Elasticsearch instances in the cluster:

```
ELASTICSEARCH.discovery.zen.ping.unicast.hosts="<esInstance1IpOrHostName>:<
esInstance1TransportTcpPort>,<esInstance2IpOrHostName>:<esInstance2Transpor
tTcpPort>,<esInstance3IpOrHostName>:<esInstance3TransportTcpPort>"
```

For example, assuming that we want to tell an Elasticsearch instance on the first worker node of our example scenario about two other instances on the second and third worker node, we will set the parameter as follows:

```
ELASTICSEARCH.discovery.zen.ping.unicast.hosts="arisworker2:14230,arisworke
r3:14230"
```

Note that the port in the list is **not** the ES client port (set with the parameter "ELASTICSEARCH.http.port", but the so-called transport port for communication between instances, which is set with the configure parameter "ELASTICSEARCH.transport.tcp.port").

> ℹ️ Due to the gossip protocol, it is sufficient to have only one existing member listed here, but because any single existing member might not be available at any given time, it is recommended that you always provide a full list of all other cluster members to each cluster member.

> ℹ️ This is very similar to how the members of a Zookeeper ensemble find each other (via the indexed "server" configuration parameter). However, for Zookeeper, each ensemble member **must** have the full list of members. Another difference is that each Zookeeper ensemble member has two server ports that must be added to the list, while Elasticsearch has only one port for communication among the cluster instances, the transport port.

5.3.1.2 Additional Elasticsearch cluster configure parameters

ELASTICSEARCH.discovery.zen.minimum_master_nodes

The parameter "ELASTICSEARCH.discovery.zen.minimum_master_nodes" defines the size of the quorum, i.e., the number of possible master-eligible Elasticsearch instances that must be available for a master election to proceed.

To avoid the split-brain problem, the parameter must always be set to values greater than (i.e., ">", **not** ">=") half of the total number of possible master nodes, (i.e., round((n +1)/2).

ELASTICSEARCH.index.number_of_replicas

The parameter "ELASTICSEARCH.index.number_of_replicas" controls the level of redundancy of your data, i.e., how many replica of a shard should be held in the cluster. Note that this setting refers to the copies of a shard in addition to the primary copy. The parameter should be set to 0 for single-node "clusters" (otherwise, a lone Elasticsearch instance will not have a second cluster instance available to hold the additional copy, leading to the cluster going into the degraded "yellow" state), and to 1 for actual clusters of more than one instance that are meant to be highly available. In theory, you could set this to even higher values (but never set it higher than the actual number of Elasticsearch instances in your cluster minus 1) for additional redundancy. For example, if you set this value to 2 (giving you a total of 3 copies of each shard, including the primary), Elasticsearch will distribute these shards in such a way that even if any two instances of the cluster fail, there is still at least one copy of each shard available in one of the remaining instances. However, having any more replicas will increase the storage overhead and the overhead to keep them in sync. Further, due to quorum-based consistency control, it makes no sense to have significantly more than half as many replicas as you have Elasticsearch instances - if you lose more than half your instances, your cluster will become inoperative anyway.

Note that it is technically possible to change the number of replicas for an index, however, you cannot do that using this configure parameter, because it only affects the number of replicas for newly created shards. I t is highly recommended that you decide about the initial number of replicas in your cluster at the beginning and keep it at that.

ELASTICSEARCH.index.number_of_shards

The parameter "ELASTICSEARCH.index.number_of_shards" controls into how many shards (i.e., physical parts that can be distributed across nodes) each index is divided. A change cannot be applied retroactively on existing indexes and will thus only take effect for indexes that are created afterwards. It should in general be left at its default value 3.

Note: Changes to this parameter will take effect only for indexes that are created afterwards, so make sure you set it properly before starting the system for the first time.

ELASTICSEARCH.gateway.expected_nodes

The parameter "ELASTICSEARCH.gateway.expected_nodes" must be set to the total number of nodes in the cluster.

ELASTICSEARCH.gateway.recover_after_nodes

The parameter "ELASTICSEARCH.gateway.recover_after_nodes" controls how many nodes must be available for the cluster to start shard recovery. It should be set to a quorum of all master nodes, (i.e., round((n +1)/2).

5.3.2   Example of a manually configured three-node Elasticsearch cluster

Continuing with our three-node example scenario, the following configure commands will create a highly-available three-node Elasticsearch cluster, with one Elasticsearch instance on each node n1, n2, and n3.

Example of a manually configured Elasticsearch cluster, in which all instances are located on one machine:

```
on n1 configure elastic_m es1 \
  ELASTICSEARCH.discovery.zen.ping.unicast.hosts =
"arisworker2:14230,arisworker3:14230" \
  ELASTICSEARCH.http.port = 14220 \
  ELASTICSEARCH.transport.tcp.port = 14230 \
  ELASTICSEARCH.node.local = false \
  ELASTICSEARCH.node.name = "node1" \
  ELASTICSEARCH.discovery.zen.minimum_master_nodes = 2 \
  ELASTICSEARCH.index.number_of_replicas = 1 \
  ELASTICSEARCH.index.number_of_shards = 3 \
  ELASTICSEARCH.gateway.expected_nodes = 3 \
  ELASTICSEARCH.gateway.recover_after_nodes = 2


on n2 configure elastic_m es2 \
  ELASTICSEARCH.discovery.zen.ping.unicast.hosts =
"arisworker1:14230,arisworker3:14230" \
  ELASTICSEARCH.http.port = 14220 \
  ELASTICSEARCH.transport.tcp.port = 14230 \
  ELASTICSEARCH.node.local = false \
  ELASTICSEARCH.node.name = "node2" \
  ELASTICSEARCH.discovery.zen.minimum_master_nodes = 2 \
  ELASTICSEARCH.index.number_of_replicas = 1 \
  ELASTICSEARCH.index.number_of_shards = 3 \
  ELASTICSEARCH.gateway.expected_nodes = 3 \
  ELASTICSEARCH.gateway.recover_after_nodes = 2


on n3 configure elastic_m es3 \
  ELASTICSEARCH.discovery.zen.ping.unicast.hosts =
"arisworker1:14230,arisworker2:14230" \
  ELASTICSEARCH.http.port = 14220 \
  ELASTICSEARCH.transport.tcp.port = 14230 \
  ELASTICSEARCH.node.local = false \
  ELASTICSEARCH.node.name = "node3" \
  ELASTICSEARCH.discovery.zen.minimum_master_nodes = 2 \
  ELASTICSEARCH.index.number_of_replicas = 1 \
  ELASTICSEARCH.index.number_of_shards = 3 \
  ELASTICSEARCH.gateway.expected_nodes = 3 \
  ELASTICSEARCH.gateway.recover_after_nodes = 2
```

> 🛈 Note that the parameters for the Elasticsearch client and transport ports (ELASTICSEARCH.http.port and ELASTICSEARCH.transport.tcp.port, respectively) have been added explicitly to the configure commands, even though we actually left them at their respective default values. This is meant to help illustrate the point that the port in the "unicast host" list is the transport port.

As you can see, it is easy to get things wrong. While some parameters will be the same for any three-node cluster, others are not. In particular, the manual maintenance of ELASTICSEARCH.discovery.zen.ping.unicast.hosts is difficult because you must adapt the parameter based on the actual hostnames (or IP addresses) of the machines to which you want to deploy your cluster. This also means that your template (i.e., the file containing all your configure commands) is not easily portable anymore and you must adapt it to the hostnames or IP addresses of the concrete environment.

5.4   Setting up an Elasticsearch cluster with the Elasticsearch manager

To make setting up an Elasticsearch cluster more convenient and less error-prone, we provide tooling inside ACC that supports this task. This functionality is summarized as "Elasticsearch manager". Conceptually similar to the already existing "Zookeeper manager", which solves the same problem for setting up a Zookeeper ensemble, it reduces the effort of setting up a cluster to just deciding on which nodes to put an elasticsearch instance (and deciding how to name the cluster).

5.4.1   Configuration

The only ACC configuration parameter added for the Elasticsearch manager is "esmgr.elasticsearch.app.type", which has to be set to the name of the app type that will be used for configuring Elasticsearch instances. If the parameter is not set, or if no app type with the name given as value for this parameter is found, the Elasticsearch manager will not work.

5.4.2   Commands

5.4.2.1   Validate elasticsearch cluster

The "validate elasticsearch cluster" command performs a validation of the current state of all Elasticsearch instances found across all nodes currently registered in ACC. The command checks if there is more than one cluster (or unclustered instance) across the nodes. Note that having more than one Elasticsearch cluster across the nodes of a single ARIS installation is not supported and will lead to a system that will not function properly.

Syntax:

```
"validate" ("es" | "elasticsearch") "cluster"
```

Example:

```
validate elasticsearch cluster
```

5.4.2.2   List elasticsearch instances

The "list elasticsearch instances" command displays an overview of all Elasticsearch instances found across all nodes currently registered in ACC. For each instance the command shows the name of the cluster to which the instance belongs and the ports occupied.

Syntax:

```
"list" ("es" | "elasticsearch") "instances"
```

Example:

```
list es instances
```

5.4.2.3   Add elasticsearch instance

The "add elasticsearch instance" command allows "staging" of the addition of a new Elasticsearch instance on the specified node (or the current node, if no node is specified) to the cluster with the specified cluster name. Optionally, an instance ID for the instance can be specified.
Note that this command does not yet perform an actual change of the Elasticsearch cluster.

Syntax:

```
<nodeSpec>? "add" ("es" | "elasticsearch") "instance"? "to" "cluster"
<identifier> ("using" "instanceId"? <instanceId>)?
```

Examples:

```
add es instance to cluster mycluster
```

Stages the addition of a new Elasticsearch instance on the current node that will be part of the cluster "mycluster".

```
on n1 add elasticsearch instance to cluster myescluster using elastic1
```

Stages the addition of a new Elasticsearch instance on node n1 that will be part of the cluster "myescluster" and will use the instance ID "elastic1".

5.4.2.4   Remove elasticsearch instance

Currently the Elasticsearch manager does not support the removal of Elasticsearch instances due to the inherent danger to customer data imposed by such an operation.
The reason is that we do not know if all indexes have been created with replication. If we deconfigure the only instance holding the sole copy of a shard, the data will be gone.

5.4.2.5   Validate elasticsearch changes

The "**validate elasticsearch changes**" command performs a validation of the currently staged changes of the Elasticsearch cluster. The command checks if the resulting cluster is highly available.

Syntax:

```
validate "pending"? ("es" | "elasticsearch") "cluster"? "changes"
```

Example:

```
validate es changes
```

5.4.2.6   Commit elasticsearch changes

The "commit elasticsearch changes" command applies the currently staged changes to the Elasticsearch cluster by running the required configure (for new instances) and reconfigure (for existing instances) commands.
Note that because reconfiguring existing instances requires a restart, the cluster might be temporarily unavailable during this operation. Further, we also observed that despite proper configuration of the cluster instances after this operation, the Elasticsearch cluster sometimes does not return to a working state by itself, but only after a shutdown and restart of the entire cluster. It is therefore not recommended to try and change a running Elasticsearch cluster.

Syntax:

```
commit "pending"? ("es" | "elasticsearch") "cluster"? "changes"
```

Example:

```
commit es changes
```

5.4.2.7   Reset elasticsearch changes

The "reset elasticsearch changes" command discards all currently staged Elasticsearch cluster changes.

Syntax:

```
("discard" | "reset") "pending"? ("es" | "elasticsearch") "cluster"?
```

Example:

```
discard es changes
```

5.4.3 Usage examples

A common use case is to create a highly available Elasticsearch cluster consisting of three instances spread across three nodes. Assuming your nodes are registered as n1, n2, and n3, run the following commands to create this cluster:

```
on n1 add es to cluster myelasticsearchcluster
on n2 add es to cluster myelasticsearchcluster
on n3 add es to cluster myelasticsearchcluster
commit es changes
```

Afterwards, start your newly created instances and you will have a working cluster that can tolerate the outage of one of its instances.

5.5 Elasticsearch in our example scenario

You can use the commands from our Usage examples section above to set up a three-node Elasticsearch cluster. You can run these commands directly in your ACC shell and/or you can add them to your template file. Assuming the commands complete successfully, your distributed example system will look as shown in figure 36 below.

FIGURE 36: THE ELASTICSEARCH CLUSTER IN OUR EXAMPLE SCENARIO
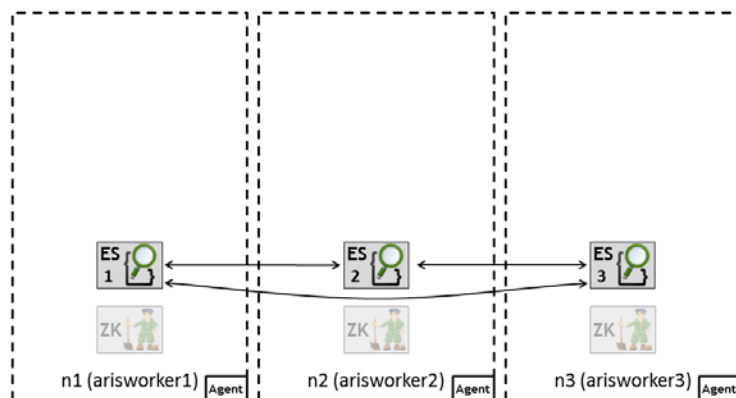
6 Relational database backends

6.1 Motivation - The role of relational databases in the ARIS architecture

Despite the adoption of some "modern" backend systems in the ARIS 9/10 architecture (like Elasticsearch or Zookeeper), the primary storage location for most of the user-generated data (ARIS databases, models, definitions; ARIS method customizations; ARIS reports; collaboration content; document metadata and content, process governance data) is still a "traditional" relational database system (RDBMS). RDBMSs have been around for decades now and are a mature technology that offers transaction semantics to ensure that any change to the data is done in an atomic, consistent, isolated, and durable way (the ACID paradigm that we briefly introduced in chapter 2).

Due to the pivotal role of the relational backend in ARIS, when aiming for a highly available, distributed ARIS installation, you must make the relational backend highly available as well.

6.2 Scaling & high availability with relational databases

The standard RDBMS shipped with ARIS is PostgreSQL, which we provide packaged as a runnable, so that it can be installed and configured using the ARIS provisioning technology. While for services like Zookeeper or Elasticsearch it is possible to configure them to form a cluster as described in the previous chapters, we do not offer an out-of-the-box clustering solution for PostgreSQL. Distributing the load of multiple tenants across several DB instances (no high availability)

The only kind of scaling that is possible with PostgreSQL is that of scaling "across" tenants. Currently, the data of a single tenant always resides in a single database instance, so the load created by a single tenant on the database instance must be able to be handled by that instance. At the same time, a tenant's database instance will also be a single point of failure for that tenant - if the tenant's DB goes offline, the tenant will become unusable until the DB is available again. However, if high availability is not relevant for an ARIS usage scenario, but if a large number of tenants must be supported, it can make sense to have multiple database instances available in the system. The data of the tenants will then be distributed (more or less) evenly across all available databases services, as illustrated in figure 37 below.

FIGURE 37: A POSSIBLE DISTRIBUTION OF TENANTS ACROSS MULTIPLE POSTGRESQL INSTANCES
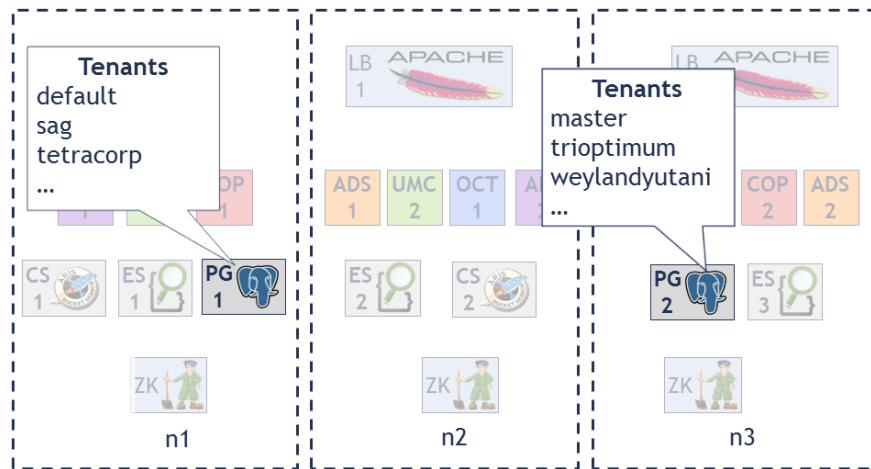
Here, we have two instances of our PostgreSQL runnable, one on node n1, another on node n3. All tenants created in this ARIS installation will be distributed evenly across these two runnables, as indicated by the callouts.

While this offers a way of spreading the load of multiple tenants across several instances, it is rarely used in practice. In particular for production use on mission-critical systems, you are usually interested in high availability, which this approach will **not** provide.

6.2.1　High availability for the relational database backend by using an external, highly available database cluster (Oracle or SQL Server)

Because we do not support a highly-available configuration using the PostgreSQL runnable, you must use an external DBMS (where "external" means "software not provided together with ARIS") like Oracle or MS SQL, which offers mechanisms for clustering and high availability. For details about setting up such a DBMS cluster, see the documentation of the external database that you plan to use.

From the perspective of ARIS, a clustered external DBMS behaves like its non-clustered counterpart, i.e., clustering is transparent to the application. Consequently, the steps needed to make ARIS use a clustered external database are the same as for setting up ARIS with a NON-clustered external DBMS (the only difference might be the way ARIS connects to the cluster, in particular the database URL can differ from that of a non-distributed external DBMS - but this information will be provided by the database administrator). The following sections will detail the steps needed for setting up ARIS with an external DBMS (or refer to other documentation for some aspects).

6.2.1.1 Registering the external DBMS as an external service

Because our runnables use Zookeeper to find services, the information needed for ARIS to find and use an external DBMS must be added to Zookeeper as well. ACC commands are used for registering and updating external services. With the *register external service* command you perform the initial registration of the service. With the *list external services* command you can get an overview of all external services that are currently registered. With the *show external service* command you can review the properties of individual registered services. With the *update external service* command you can change individual properties of the registered service.

Note that for these commands to work, you must start your Zookeeper runnable (or Zookeeper runnables, if you have a highly available Zookeeper ensemble).

Registering an external Oracle DBMS

To register an external Oracle DBMS, you run the "register external service" command structured like this (note that the command below has been broken up across several lines by adding "\" character in front of every line break that we introduced for legibility).

```
register external service db \
 host=<OracleHostnameOrIP> \
url="jdbc:oracle:thin:@<OracleHostnameOrIP>:<OraclePort>/<OracleServiceName
>" \
 driverClassName=oracle.jdbc.OracleDriver \
 jmxEnabled=true \
 username="<AppUserName>" \
 password="<AppUserPwd>" \
 maxIdle=15 \
 maxActive=300 \
 maxWait=10000 \
 removeAbandoned=false \
 removeAbandonedTimeout=600 \
 logAbandoned=false \
 initialSize=0 \
 testOnBorrow=true \
 validationQuery="select 1 from dual" \
 defaultAutoCommit=false \
 rollbackOnReturn=true \
 jmxEnabled=true
```

Replace the placeholders (the parts in angled brackets), i.e.,<OracleHostnameOrIP>, <OraclePort>, <OracleServiceName>, <AppUserName>, and <AppUserPwd>, with the hostname (or the IP address), the port and the service name of your Oracle DBMS, the name of the application user and the password of this user (as you specified them in the envset.bat file that comes as a part of the schema creation scripts for the variables CIP_APP_USER and CIP_APP_PWD, respectively), respectively. Note that the <OracleHostnameOrIP> placeholder is used twice, once for the value of the "host" parameter, and a second time as part of the value of the "url" parameter.

The values for <OracleHostnameOrIP>, <OraclePort>, <OracleServiceName> (or the entire value of the "url" parameter) must be provided by your database administrator.

> ℹ️ You will observe that many of these parameters hold values that do not pertain in any way to the actual details of the database system being used. These parameters control the usage of the DBMS by the application, in particular the handling of the JDBC connection pool. These parameters can be used in demanding scenarios to tune the database usage to specific use case. For most scenarios, however, the default values you can find in the command above are the only ones that you need. Therefore, unless explicitly told by ARIS R&D to do so you must not change these values from the defaults shown above.

For example, assuming that your Oracle DBMS is running on the machine myoracle.example.com on port 1521, uses the service name XE, and you used the application user name "arisuser" and password "TopS3cret", the actual command is:

```
register external service db \
  host=myoracle.example.com \
  url="jdbc:oracle:thin:@myoracle.example.com:1521:xe" \
  driverClassName=oracle.jdbc.OracleDriver \
  jmxEnabled=true \
  username="arisuser" \
  password="TopS3cret" \
  maxIdle=15 \
  maxActive=300 \
  maxWait=10000 \
  removeAbandoned=false \
  removeAbandonedTimeout=600 \
  logAbandoned=false \
  initialSize=0 \
  testOnBorrow=true \
  validationQuery="select 1 from dual" \
  defaultAutoCommit=false \
  rollbackOnReturn=true \
  jmxEnabled=true
```

Assuming that the command was executed successfully, it will return the technical ID given to the newly registered external service in Zookeeper:

```
ACC+ >register external service db host=myoracle.example.com [...]
New external service of type db registered with serviceId db0000000000.
```

Notice the service ID (here "db0000000000").
You will later need this ID when we assign the ARIS tenants to the database system, as detailed in the section below. You can note it down now, or you can look it up later using the aforementioned "list external services" command.

Registering an external MS SQL DBMS

The procedure to register an external Microsoft SQL Server DBMS is similar to that for Oracle. The only difference is the default (and fixed) value of the "driverClassName" parameter and the structure of the value of the "url" parameter. The basic structure of the "register external service" command for MS SQL is the following:

```
register external service db \
 host=<SqlServerHostnameOrIP> \

url="jdbc:sqlserver://<SqlServerHostnameOrIP>:<SqlServerPort>;DatabaseName=
<DatabaseName>" \
 driverClassName=com.microsoft.sqlserver.jdbc.SQLServerDriver \
 jmxEnabled=true \
 username="<AppUserName>" \
 password="<AppUserPwd>" \
 maxIdle=15 \
 maxActive=300 \
 maxWait=10000 \
 removeAbandoned=false \
 removeAbandonedTimeout=600 \
 logAbandoned=false \
 initialSize=0 \
 testOnBorrow=true \
 validationQuery="select 1 from dual" \
 defaultAutoCommit=false \
 rollbackOnReturn=true \
 jmxEnabled=true
```

Replace the parts in angled brackets, i.e., <SqlServerHostnameOrIP>,<SqlServerPort>, <DatabaseName>, <AppUserName>, and <AppUserPwd> with the hostname (or IP address), the port, the database name and the database instance name of your SQLServer DBMS, and the application user name and password as you set them in the envset.bat before running the schema creation scripts. Note that the <SqlServerHostnameOrIP> placeholder is used twice, once for the value of the "host" parameter, and a second time as part of the value of the "url" parameter.

For example, assuming that your MS SQL Server DBMS is running on the machine mysqlserver.example.com on port 1433 and the database name is "arisdb", and assuming that the username and password of the application user are "arisuser" and "TopS3cret", respectively, the actual command is:

```
register external service db \
  host=mysqlserver.example.com \
  url="jdbc:sqlserver://mysqlserver.example.com:1433;DatabaseName=arisdb" \
  driverClassName=com.microsoft.sqlserver.jdbc.SQLServerDriver \
  jmxEnabled=true \
  username="arisuser" \
  password="TopS3cret" \
  maxIdle=15 \
  maxActive=300 \
  maxWait=10000 \
  removeAbandoned=false \
  removeAbandonedTimeout=600 \
  logAbandoned=false \
  initialSize=0 \
  testOnBorrow=true \
  validationQuery="select 1 from dual" \
  defaultAutoCommit=false \
  rollbackOnReturn=true \
  jmxEnabled=true
```

As was described in the section before about the registration of an Oracle DBMS, you must note the service Id given to your newly registered database service.

6.2.1.2   Creating the database schemas for each tenant

ARIS is a multi-tenant-enabled application, i.e., a single physical ARIS installation (whether it is a single-node or multi-node, distributed installation) can host several "logical ARIS installations", known as "tenants". The different tenants share the infrastructure of their installation, but have completely separated content (such as users, ARIS databases, collaboration content, documents, process governance processes, etc.). Inside the DBMS the data of different tenants is separated by holding it in different database schemas (or their respective equivalent;Oracle DBMS does not have the concept of a schema as a "container for database tables", but uses databases users for that purpose). Even if you do not plan to use more than one tenant for actual ARIS project work, each ARIS installation has **at minimum two tenants**, named "default" and "master". "default" is the tenant that by default is used for ARIS project work. "master" is the tenant used for managing other tenants, in particular for creating new tenants and deleting existing tenants, or for access to the ARIS health check UI.

Usually, the administrators of an Oracle or MS SQL database server will not provide an application with the necessary access rights so that it can create its own database schemas. Therefore, the administrator will need to run a set of scripts provided by ARIS (at the time of this writing these scripts are located on the ARIS DVD at the location Add-ons\DatabaseScripts) that create the required schema objects with the proper permissions. The preparation and usage of these scripts is described for the different database systems and installation platforms in the "ARIS Server Installation guides" found on the ARIS DVD (at the time of this writing located at Documents\English\3 Installation\31 Initial installation\312 Server, files "ARIS Server Installation - Linux.pdf" and "ARIS Server Installation - Windows.pdf"). In general, a dedicated schema should be created for each tenant. Because each ARIS installation has at minimum two tenants ("default" and "master"), you must create at least two schemas, plus additional schemas for each additional tenant you want use on the installation (additional tenants and their associated schemas can be added at a later stage, as described in section 1.18 of the document "ARIS Cloud Controller (ACC) Command-line Tool".

> ℹ In ARIS 9.x the database scripts for Oracle not only created the actual database schema object (i.e., the "container for tables") and the required roles and permissions, but also created the actual database tables in that schema. With ARIS 10, the database schema is created in such a way that the application has the necessary permissions to create the database tables itself automatically. While this makes little difference for a fresh installation, it makes updating an ARIS 10 installation more convenient because it also makes it possible to handle any schema update operations by the application (for example, adding new tables or adding new columns to existing tables).
>
> So, different from ARIS 9.x, where schema update scripts had to be executed when updating an ARIS installation that was using Oracle as an external DBMS, this is no longer necessary when updating from ARIS 10.x to ARIS 10.y.

6.2.1.3   Assigning the tenant to the database service

After you registered your Oracle or MS SQL Server database system with ARIS and created the schemas for each tenant, you must tell ARIS two things:

1. The database service that should be used for each tenant.
2. The name of the schema in the respective database that should be used for each tenant.

Point 1) is needed because when an external DB service is used, ARIS will not automatically assign tenants to it (as it does when our PostgresSQL runnable is used), and because there can be multiple database services, as explained in the section about horizontal scaling across tenants above. Point 2) is needed because the choice of naming the schemas for the different ARIS tenants is at the discretion of the database administrators and will in general be different from the default schema naming rules used when ARIS itself can create the schemas (i.e., when our PostgresSQL runnable is used).

Fortunately, both aspects can be handled with a single ACC command that you run once for each tenant, the "assign tenant to service" command. The command creates a logical connection between a tenant and a concrete service instance (in this case an instance of the service type DB, as we created it in Zookeeper when we registered the external DB above). In addition, parameters can be specified for this logical connection. The only mandatory parameter is "com.aris.cip.db.schema", which is used for specifying the schema name.

The basic grammar of the "assign tenant to service" command is

```
"assign" "tenant" <tenantId> "to" "service" <serviceId> (<key> "=" <value>)*
```

Here, <tenantId> is the ID of the tenant - "default" and "master" for the two tenants that implicitly exist in every ARIS installation, or any string of a maximum length of 30 characters that starts with a lower-case Latin letter (i.e., "a" - "z") and consists of only lower-case Latin letters and digits; <serviceId> is the ID of the service as it was returned by the "register external service command" above. <key> and <value> are the name and the value of various properties that can be stored with the tenant-2-service assignment. Multiple such key-value pairs can be specified.

For example, assuming your registered external DB service was given the service ID db0000000000 and you named the schema for the master tenant "aris_master" and the schema for the default tenant "aris_default", you run the following two commands to make ARIS use this DB and these schemas for these tenants:

```
assign tenant master to service db0000000000
com.aris.cip.db.schema="aris_master"

assign tenant default to service db0000000000
com.aris.cip.db.schema="aris_default"
```

> ⓘ Note that when creating additional tenants for an ARIS installation that uses an external DBMS service, you must run the "assign tenant to service" command **and** create the corresponding schema **before** running the "create tenant" command.
> This is detailed in section 1.12 of the "ARIS Cloud Controller (ACC) Command-line Tool" manual.

6.2.1.4 Enhancing the runnables with the JDBC driver for the external database system

You must add the JDBC drivers to all runnables that access the database (ADSAdmin, ABS, APG, Octopus, Simulation, Dashboarding, ECP, Copernicus) by using the Provisioning enhancement functionality, using the commonsClasspath enhancement point. The basic structure of the required enhancement command is

```
on <nodeName> enhance <instanceId> with commonsClasspath path
<pathToJdbcDriverFileInRepo>
```

By default, the file with which you enhance the runnable is taken from the same repository as the runnables themselves. The simplest way to make the JDBC driver available for the enhancement commands is to copy the file into a subdirectory of the remote repository that you are using to configure your runnables (as described in chapter 3, where we presented the preparational steps required before you can start setting up a distributed system).

Let's assume your JDBC driver JAR file was named driver.jar and you copied it into the subdirectory "jdbc" of your repository server's root directory. To enhance a runnable named "abs1" on node n1 with this file, you run the command

```
on n1 enhance abs1 with commonsClasspath path "jdbc/driver.jar"
```

> ⓘ While it is not necessary to put the path to the driver JAR file in quotes in this example because it does not contain any blanks or special characters, it is good practice to always use quotes here.

Once we configure all application runnables of our distributed system as it will be described over the course of the next sections, we will run this enhancement on all runnable instances of the aforementioned types over all nodes.

6.3   Relational databases in our example scenario

After configuring the external DB for use with ARIS as detailed above, our distributed example scenario looks as shown in figure 38 below.

FIGURE 38: OUR DISTRIBUTED EXAMPLE SCENARIO AFTER REGISTERING A HIGHLY-AVAILABLE EXTERNAL DATABASE SYSTEM
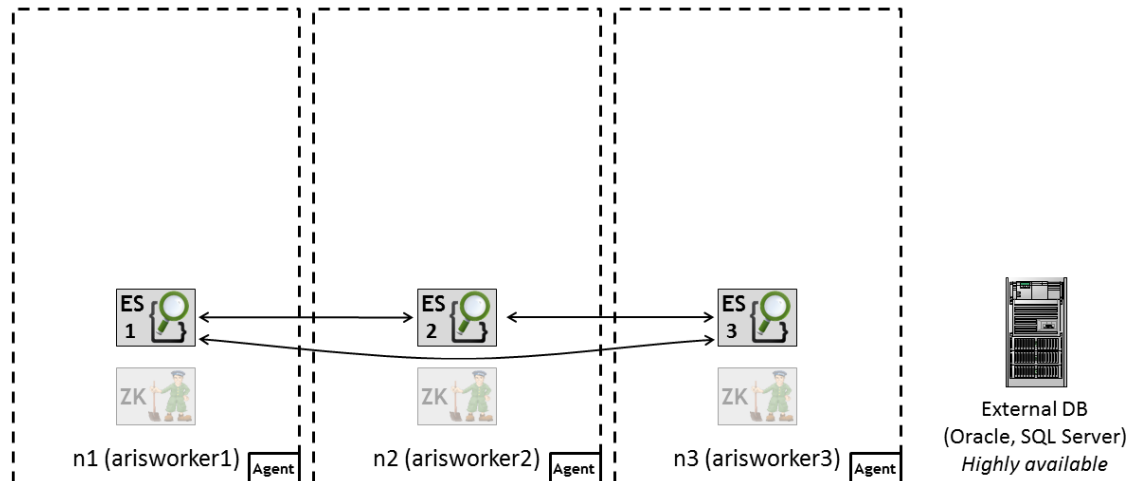
Note that you cannot yet run the above commands at this point (and you also should not add these commands to the same template file that hold the "configure" and related commands). Since the "register external service" and "assign tenant" commands require a running Zookeeper cluster, you should run them after all runnables are configured (or add them to a dedicated "template" file).

If your goal is to make yourself familiar with distributed installations in general, but if you want to avoid setting up an external DBMS for use with ARIS for now, you can just configure a single instance of the PostgreSQL runnable on one of your nodes (or an additional node), as indicated in figure 39 below. This means however that this runnable and this node will be a single point of failure for your installation, so this should be used for "exercise only":

FIGURE 39: OUR DISTRIBUTED EXAMPLE SCENARIO WHEN USING THE POSTGRESQL RUNNABLE INSTEAD (I.E., NO HIGH AVAILABILITY, STRICTLY EXERCISE ONLY)
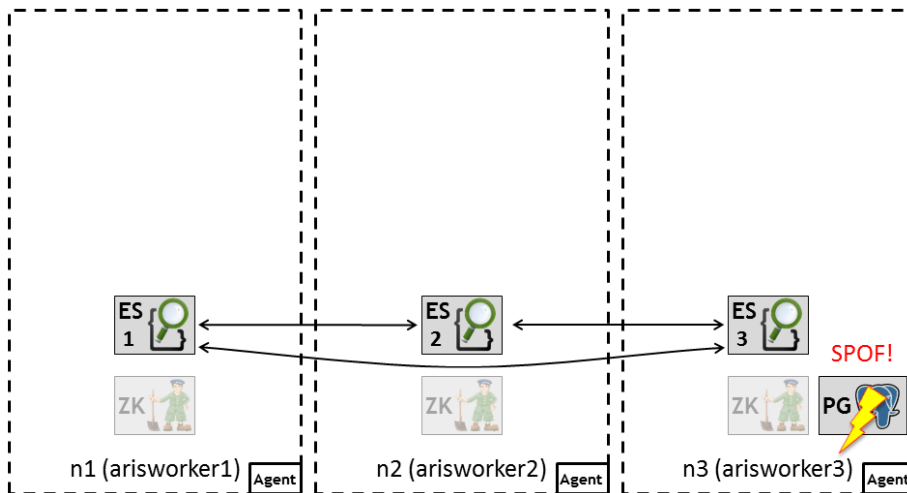
To do this, you must run a configure command with a postgres app type, for example, to configure your PostgreSQL runnable instance on node n3 as shown above and name it "pg", you run the command

```
on n3 configure postgres_m pg
```

(and/or add it to your template file).

7   Cloudsearch clustering

7.1   Motivation - The role of Cloudsearch in the ARIS architecture

> **ⓘ Disclaimer**
>
> The following high-level description of the motivation for introducing the Cloudsearch component to the ARIS architecture requires understanding of relational databases.

One major change in the use of relational databases from ARIS 7.x to ARIS 9/10 is that the underlying relational schema has changed significantly. In ARIS 7.x, complex ARIS entities (like models or object definitions) are stored in a traditional "third normal form" (3NF) schema, where all properties of an entity are split up into individual columns and (for multi-valued properties) into dedicated tables connected with foreign-key relationships (just imagine that your complex object is first disassembled and then all parts are stored). While this traditional mapping of complex entities to the relational data model has its benefits (in particular, you can access any property of an entity directly with SQL, allowing you to express virtually any kind of query), the drawback is that in order to load such an entity from the database, it must be reassembled by many, often complex queries across many database tables, which is often slow and puts a significant load on the DBMS. And since for most use cases, the complex entity is loaded as a whole and rarely only as individual parts, this structuring of the data made the most common use cases slow.

Starting with ARIS 9, complex entities are stored in an efficient, compressed binary form (as "binary large object", or "BLOB" columns). Thus, loading a complete entity is a single key-based database access, which is very fast. The drawback is, however, that it is no longer possible to perform complex queries on ARIS content via SQL. This drawback is overcome by using Cloudsearch: Cloudsearch holds indexes over the ARIS data, both indexes for attribute-based (including full-text) search and structural indexes that describe the relationships among ARIS entities. Cloudsearch indexes are held completely in memory, making it possible to answer even complex queries (that would correspond to large joins across many tables) without any disk access. With Cloudsearch indexes, it is feasible to perform even complex structural queries over ARIS content with good response times and with high throughput, which is needed in particular for the ARIS Connect portal, which is built to allow a large number of concurrent users (primarily read-only) access to the ARIS repository.

The idea is to leave the aspect of securely storing the valuable customer data where it is done best (i.e., in the mature RDBMS), while the handling of complex queries specific to ARIS is done in a dedicated, highly optimized component.

How are the database content and the Cloudsearch indexes kept in sync with each other when you change something?

CIP is the component through which all database access of applications using Cloudsearch is handled. Whenever a change is made, the change is written to the database (e.g., if you save a model, that model's BLOB is written to the DB). Further, all updates to the Cloudsearch indexes that result from this change are sent synchronously to Cloudsearch (or

rather to all Cloudsearch instances holding the relevant index, more on that later), which will then apply the changes to its **in-memory** index structures (i.e., the change is not necessarily persisted). While the synchronous communication to the Cloudsearch instances incurs some overhead and also means that the slowest Cloudsearch instance is the limiting factor for the response time of a change operation, this approach ensures that all Cloudsearch instances immediately have all changes, so that a query sent directly after a change will directly see the change (i.e., from the perspective of the application Cloudsearch content is always fully consistent with the database content). In addition, all index changes are stored in the database as so-called "pending packets". This is needed to make sure that changes to the Cloudsearch indexes are not lost if a Cloudsearch instance becomes unavailable. To ensure good performance for updates, the index changes in Cloudsearch are **not** persisted to disk immediately (as otherwise any change operation would synchronously write to the database and to the disk of all Cloudsearch instances, which is very slow) but **only** updated in memory. A background job in Cloudsearch persists the changes from time to time in a batch operation to the Cloudsearch index files on disk. Only after a change has been written to disk by all Cloudsearch instances holding the relevant index(es), can the "pending packet" stored in the database be safely deleted. This mechanism ensures that the Cloudsearch indexes are always in sync or in the case of a crashed Cloudsearch instance can easily be made to be in sync by "replaying" the remaining "pending packets" to the restarting Cloudsearch instance.

It is important to point out that the database still holds all available information about the ARIS content. The data held in the index structures in Cloudsearch is derived from the database content. This has the advantage that if for some reason the Cloudsearch indexes and the database content become out of sync or the index is corrupted in some way, it can be rebuilt from the database content, a process known as re-indexing. Depending on the amount of ARIS content, this process can take a while, but your data is always safely stored in the database.

> **ℹ Which applications use Cloudsearch?**
>
> Note that not all ARIS components make use of Cloudsearch and the CIP module. In ARIS 9.8.x, the business server (ABS), the portal (Copernicus), analysis (Octopus), and process governance (APG) use CIP. In versions before 9.7.x, also user management (UMC) data was stored via CIP. The Collaboration component (ECP) uses a traditional 3NF schema and for that reason had its own database instance, "PostgreSQL-ECP" in ARIS 9.x.
>
> In ARIS 10, only the business server (ABS), portal (Copernicus), and analysis (Octopus) components use CIP and Cloudsearch, which have been further optimized to cover their specific use cases. APG (like ECP before) now also uses a more traditional 3NF schema, as does the document storage component (ADS), whose backend was changed from 9.x to 10 to use the database for the storage of document metadata (and optionally also document content). Despite the fundamental difference in how CIP-based and non-CIP-based applications represent their data in the database, with ARIS 10 all applications can share a single database system (and even share a single schema for each tenant), which is why the dedicated database system for Collaboration has been removed for ARIS 10.

7.2 Cloudsearch clustering, variant 1: data load distribution of multiple tenants across several instances, no high availability, no query load distribution

In a standard single-node installation, a single Cloudsearch holds the index data of all tenants in the installation.

When you add several Cloudsearch instances to the system, a tenant's Cloudsearch index data is handled similarly to how it is handled when having multiple database systems in an installation(as we discussed it in chapter 6): all data of an individual tenant is stored in a single Cloudsearch instance. If multiple Cloudsearch instances are available, the data of different tenants is distributed across the available instances, as illustrated in figure 40 below:

FIGURE 40: TWO CLOUDSEARCH INSTANCES WITHOUT REDUNDANCY
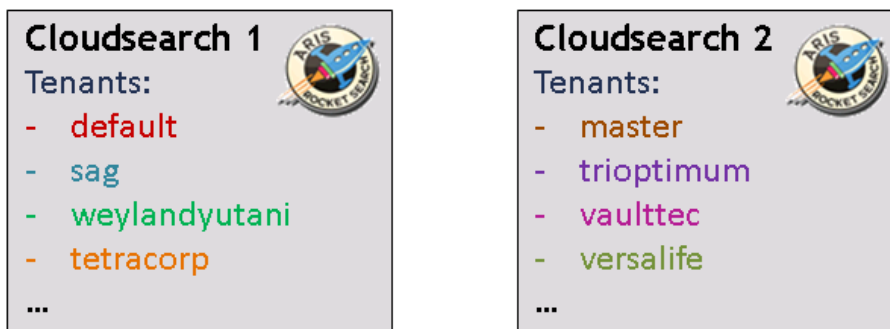
Observe that in this configuration, using two Cloudsearch instances in your system **only allows the distribution of the data and load caused by different tenants** across different instances - it **does not allow spreading the query load of a single tenant** across multiple instances, **nor does it offer high availability**. When a Cloudsearch instance becomes unavailable, the tenants whose data resided on that instance are not able to use any ARIS components that require Cloudsearch access (in particular the business server and the portal) until the issue is resolved.

7.3   About failure scopes

To make a set of Cloudsearch instances highly available, you must configure the instances appropriately by setting the data center parameter (**zookeeper.application.instance.data center**, abbreviated to **data center** in all figures below) on all instances. The idea is that all Cloudsearch instances that have the same value for their data center parameter are considered to be located in the same "failure scope" (a term that we introduced in chapter 2 of this document), i.e. an environment that is likely to fail entirely. In the simplest cases, a "failure scope" is an individual worker node - if that node fails, then all instances on that node fail at the same time. In a larger deployment, the failure scope could be an individual host or hypervisor (i.e., rather the physical server machine running several VMs, some of which serving as ARIS workers), or it could be all physical servers located in the same server rack. If the hypervisor or the hardware it runs on fails, all VMs fail at the same time. Similarly, if the server rack has, e.g., a power supply or network issue, all machines in that rack fail. In the most ambitious deployments, you might want to spread ARIS across different data centers, so here an entire data center would be a "failure scope" and this is also where the configure parameter got its name from.

7.4 Cloudsearch clustering, variant 2: high availability and query load distribution, no data load distribution

To make Cloudsearch highly available, more than one Cloudsearch instance must exist in the system and there must be instances with different values for the "zookeeper.application.instance.data center" parameter, i.e., instances allocated to different data centers = failure scopes. In general, Cloudsearch and the CIP module allocate the data of each tenant to exactly one instance per data center (i.e., you will have as many identical copies of each tenant's index data in the system as you have different values for the data center parameter). If, for example, you have two Cloudsearch instances and declare one to be in data center A and the other to be in data center B, as shown in figure 41 below, both instances will hold the full index data of all tenants in the system:

FIGURE 41: TWO CLOUDSEARCH INSTANCES ASSIGNED TO TWO DIFFERENT DATA CENTERS
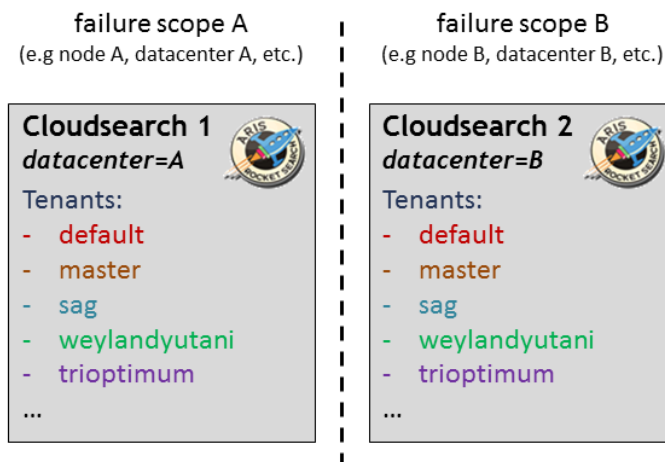
This redundancy will allow the system to continue working if one Cloudsearch instance (or the entire failure scope in which it is running, i.e., the node, host, rack, or data center) fails, i.e., it **offers high availability**. At the same time, as long as both instances are available, **query load (but not update load) of individual tenants can also be distributed** across the two instances.

A disadvantage of this variant, as is usual with high availability, is that it comes at **the price of increased resource usage** as you need double the amount of CPUs and memory for the additional Cloudsearch instances. This is necessary because **each Cloudsearch instance must hold the full amount of data** of all tenants. Remember that a key element of Cloudsearch query performance is that it can hold the data in memory. If you have many tenants and/or a great amount of data in your system (e.g., large ARIS databases), you will soon have to increase the amount of RAM available to your Cloudsearch instances (by increasing the Java maximum heap size using the "JAVA-Xmx" parameter), up to the point that you will not have enough RAM on a single node, or to a point that the heap size becomes so large that the Java garbage collection mechanisms become inefficient, leading to degrading performance due to increasingly long pauses for full garbage collection.

7.5 Cloudsearch clustering, variant 3: high availability, query and data load distribution

To combine the advantages of clustering variants 1 and 2, you should have both instances in different data centers and more than one instance per data center. Figure 42 below shows a minimal configuration of this variant, with two instances in data center A, and another two in data center B.

FIGURE 42: FOUR CLOUDSEARCH INSTANCES ASSIGNED TO TWO DIFFERENT DATA CENTERS
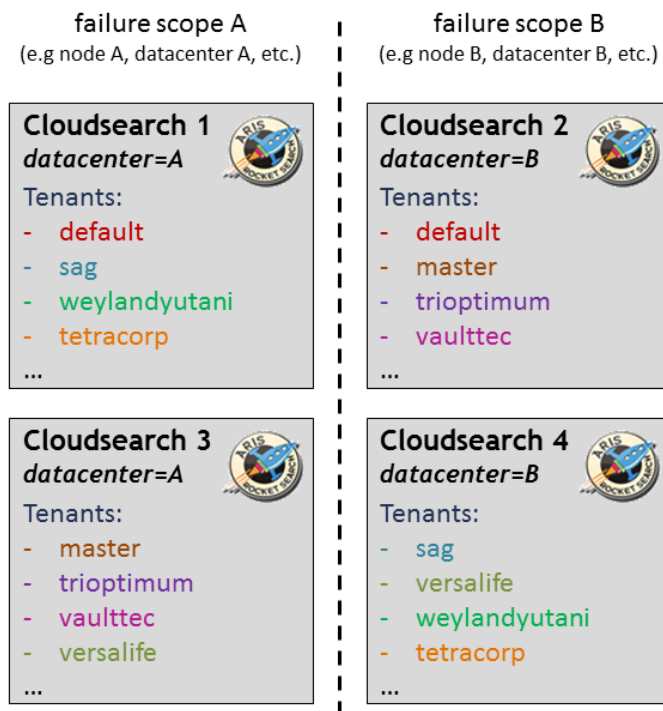
Observe how the index data of each tenant is found twice in the system, and once in each data center: for example, the data of the tenant "sag" is found on Cloudsearch instance 1 in data center A and on instance 4 in data center B. This redundancy again **offers high availability**, in that the system **can tolerate the outage of an entire failure scope** because all data will still be available in the surviving failure scope.

Note that while in this configuration the system can tolerate the outage of the two Cloudsearch instances that are in the same failure scope, this system can**not** tolerate the outage of any two instances. If. for example. Cloudsearch instances 1 and 2 fail, there is no surviving instance that is allocated to handle the data of the default tenant.

While the guaranteed maximum number of failing instances that the system can survive has not increased compared to variant 2, we now **distribute the query load of individual tenants across two instances** (one instance in each data center) and we **distribute the total amount of data (i.e. of all tenants) across two instances** in the same data center. As with variant 2 before, variant 3 comes at **the price of increased resource usage**

because we need double the amount of CPUs and memory for the additional Cloudsearch instances.

Depending on the number of tenants and the total amount of data in your system, you can scale variant 3 in two ways. You can increase the number of failure scopes, i.e., have more data centers. If you add another two instances in a data center C, these two instances together will again hold replicas of the data of all tenants, as shown in figure 43 below.

FIGURE 43: THREE DATA CENTERS WITH TWO CLOUDSEARCH INSTANCES IN EACH
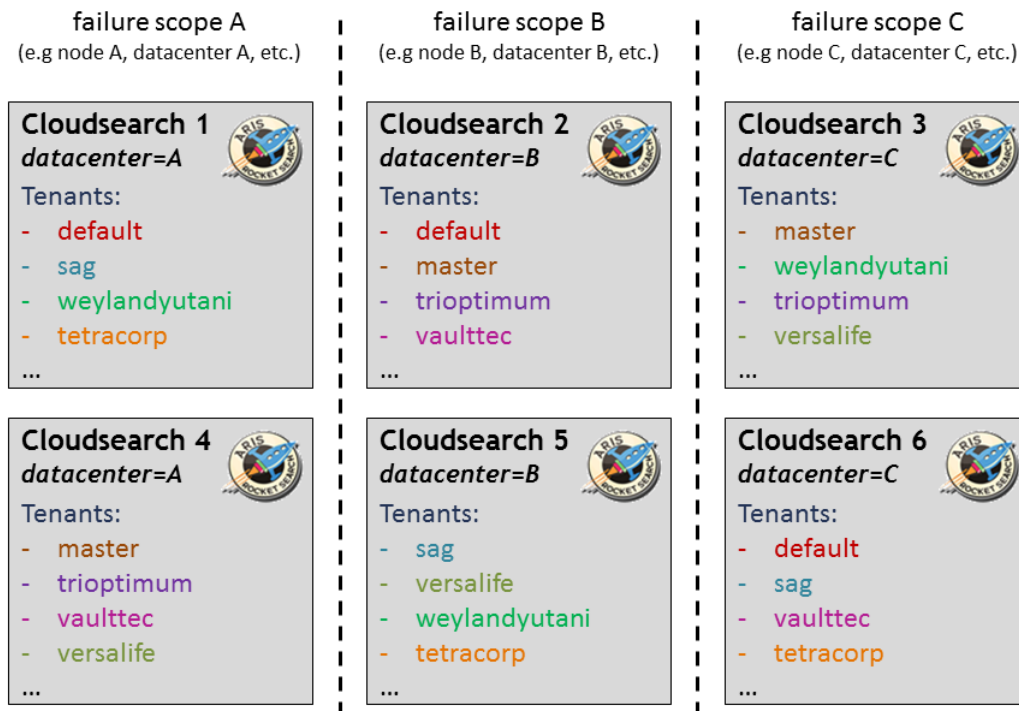
Consequently, if you set up your system this way, the **system can tolerate the outage of two entire failure scopes** (or the failure of any two instances regardless of which failure scope they are in). Because we now have three replicas of each tenant's data in the system, we tripled the resource requirements.

Alternatively, instead of having six instances spread over three data centers, you can also have only two data centers with three instances each, as illustrated in figure 44. Here, the load of the tenants is spread across more instances, but because you now have only two replicas of each tenant's data, your system can tolerate the outage of one failure scope.

FIGURE 44: THREE DATA CENTERS WITH THREE CLOUDSEARCH INSTANCES IN EACH
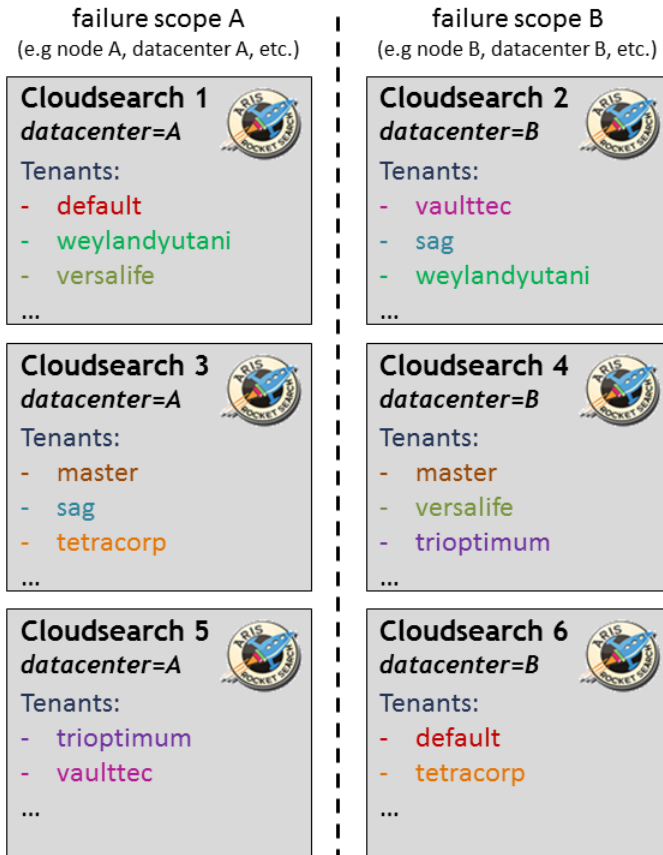
7.6 Some general guidelines & recommendations

The different Cloudsearch deployment scenarios sketched above are meant only as examples to give you an understanding of how a different number of instances distributed across a different number of data centers yield varying levels of redundancy (and thus failure tolerance), query, and data load distribution. You can adjust these scenarios according to your requirements. This section contains general guidelines to aid your Cloudsearch clustering decision.

7.6.1 Multiple instances in one data center only make sense when using more than one tenant

If you are effectively using only one tenant in your system (as is the case in many on-premise scenarios where many customers will work only with the default tenant), it makes little sense to have more than one Cloudsearch instance per data center. There are always at least two tenants in the system (master and default), and if there are two instances per data center, the CIP module puts the default tenant on one instance and the master tenant on the other instance in each data center. However, because the master tenant does not contain any modeling data, it makes little sense to dedicate a complete Cloudsearch instance to it, which will be basically idle, as illustrated in figure 45 below.

FIGURE 45: ONE ACTIVE TENANT AND TWO CLOUDSEARCH INSTANCES
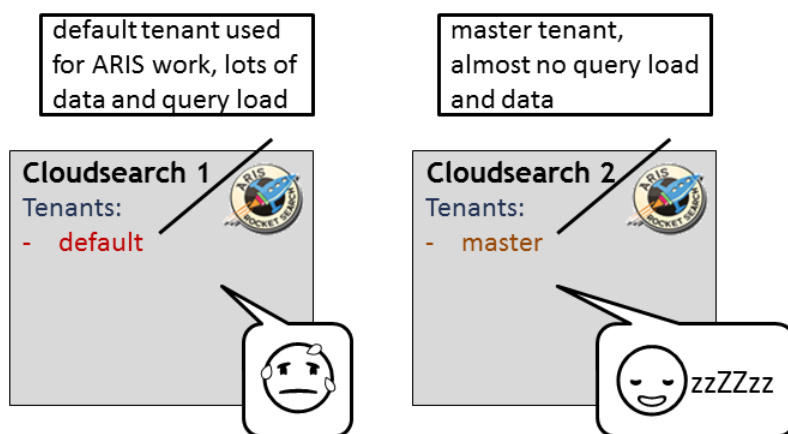
Here the basic scenario described in variant 2 above is completely sufficient. You can add additional data centers to increase redundancy or distribute query (but not data) load) and dedicate the resources to increase the sizing of the single Cloudsearch instance in each data center (in particular increasing its Java heap).

7.6.2   Size your Cloudsearch instances equally

A general rule is that all Cloudsearch instances should be sized equally (i.e., they should be given the same amount of memory using the Java maximum heap size setting, "JAVA-Xmx") and should be running on identical hardware.

This is of particular importance as this is what the load distribution mechanisms assume. For example, if you have two instances in one data center and one has less resources than the other, CIP will still assume them to be sized equally and will thus put the same number of tenants (and thus the same amount of data, assuming that tenants are sized equally) on each. Having the same amount of equally performing CPU cores for each instance is also important because all change operations must be done synchronously in all instances assigned to the respective tenant. If one instance is significantly slower, it will be the limiting factor.

7.6.3   Have the same number of Cloudsearch instances in each data center

Not only should your instances be of the same "size", but you should also make sure that you have the same number of instances in each data center. Otherwise, the instances in the data center that have fewer total instances will on average carry more load, leading to on average slower response time, again making them the limiting factor, in particular for the synchronous index update operations.

7.6.4   Adding Cloudsearch instances retroactively will not help existing tenants

Adding new instances in the same data center will not make the CIP module automatically redistribute the indexes of existing tenants to the new instances to balance the load due to the high cost of such an operation. Only new tenants created after the new instance(s) has/have been added will be found in these new instance(s), up to the point that the number of tenants is again roughly equal among all instances.

If you have several tenants and only one Cloudsearch instance, and the Cloudsearch performance or memory usage is becoming the bottleneck, it is not enough to add another instance to the same data center. If you want to redistribute the tenants among the Cloudsearch instances, the simplest approach is to stop the system, deconfigure *all* Cloudsearch instances, and configure them freshly in the new configuration. That way, CIP can make the tenant-to-Cloudsearch assignment from scratch and then distribute the tenants equally among all instances of each data center. This requires re-indexing of all Cloudsearch data, which can take time.

105

7.7   Cloudsearch in our example scenario

For our example scenario, let us assume that we have only one or a few tenants (so that the data of all tenants can still be held in a single Cloudsearch instance), but we are interested in making Cloudsearch highly available. So we will need to have at least two instances of Cloudsearch in different failure scopes. In our small three-node scenario, we will consider each node as its own failure scope (or "mini data center"). Since we configured our other services (Zookeeper and Elasticsearch) so that we can survive the outage of at most one instance (or one node, i.e. one "mini data center"), it makes little sense to have three Cloudsearch instances distributed across three data centers (which would give the Cloudsearch service as a whole the ability to survive two outages), so we stick with two Cloudsearch instances, one on node n1, the other on node n3, as shown in figure 46 below. We consequently use the node name as the name of the data center. When we later distribute the micro-service/application runnables to our example scenario, we are going to add a few more runnables to node n2 to make up for the imbalance we now created.

FIGURE 46: CLOUDSEARCH IN OUR EXAMPLE SCENARIO
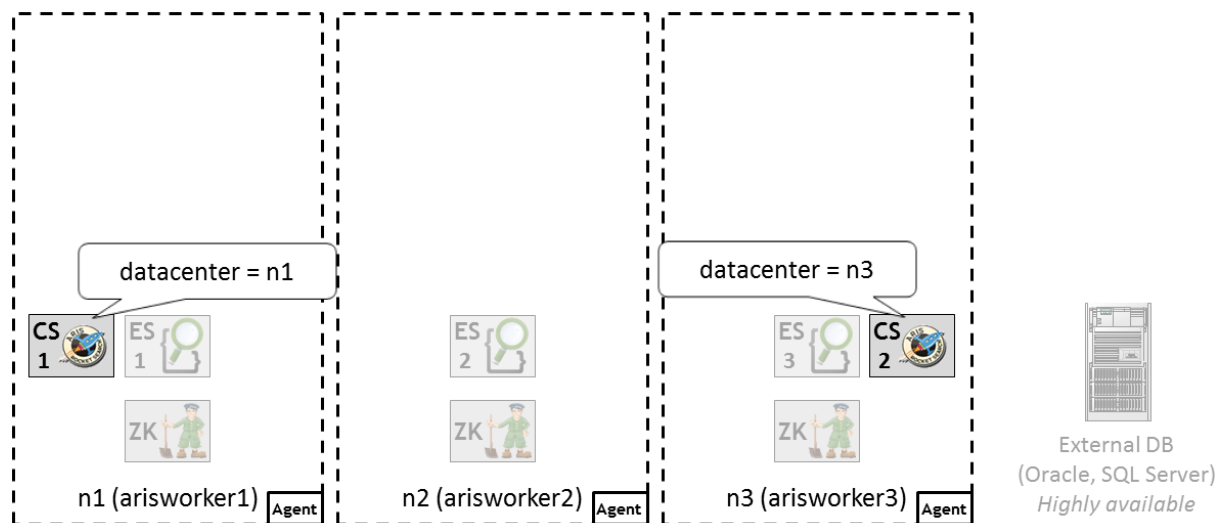
To configure these two new instances, run the commands

```
on n1 configure cloudsearch_m cs1 zookeeper.application.instance.datacenter = n1
on n3 configure cloudsearch_m cs2 zookeeper.application.instance.datacenter = n3
```

(and/or add them to your template file).

8 Clustering ARIS micro-service-based applications

8.1 Motivation - How distributing micro-service-based applications works

With a few minor exceptions, the ARIS applications or micro-services are fully stateless. To make them highly available, you must configure more than one instance of each type of application required for the actual product. Depending on how you distribute the application across nodes or data centers, you can protect yourself against the outage of a single runnable (if all instances are on one node), the outage of a node (if the instances are not all on the same node), or the outage of a data center (if the instances are distributed on nodes in different data centers).

8.1.1 Load balancing and preferred routing

The ARIS load balancer distributes the client requests for a specific type of service among all currently available instances, spreading the load and allowing the system to handle more requests.

To improve performance, for many applications we have a **preferred routing** mechanism in place, where the load balancer ensures that requests made by the same client are preferably sent to the same application instances. This allows caching in the applications to work more efficiently. To illustrate this, assume you have three instance of an application, and there is no preferred routing in place. Further, assume that requests are distributed round-robin, so every n-th request goes to instance n mod 3. (i.e., the 1st request goes to the 1st instance, the 2nd to the 2nd, the 3rd to the 3rd, the 4rd again to the 1st, and so on). The first request requires that the client's data be loaded from the backends into the caches of the first instance. The next request probably requires most of the same data, but the second instance does not have it in the cache yet, and therefore again must go to the backends to get it and store it in its own cache, which is slower than if the request is served directly from the cache (as instance 1 could do). The third request, going to the third instance, again finds the caches not containing the needed data, and again the backend must be accessed. This increases the load on the backends and slows more initial requests (until all instances have the required data in their caches). So instead of having one cache miss on the initial access to a certain data item, you will have n cache misses for n application instances. But not only that: the effective size of the caches over all application instances is reduced, as in general each instance must hold all data of all currently active clients. So the caches of the individual applications will fill more quickly, and thus cached items will be evicted from the cache more often to free space, often only to be loaded again moments later when the next request for that client comes in. This will further reduce the cache hit ratio and increase the load on the backends, possibly massively reducing the system's throughput (i.e., the total number of requests that can be handled in a given time interval) and increasing response times. With preferred routing, the load balancer, if possible, uses the same instance to answer the request of one client. These requests are thus much more likely to find the required data

already in the cache (this is usually referred to as the "the cache is hot") and can be answered much faster and with less load for the backends.

> ℹ️ **How preferred routing works**
>
> The preferred routing mechanism uses cookies, as illustrated in the screenshot below. The cookie names all start with "routing.", followed by the application type (e.g., "abs", "ecp", etc.). The value of the cookie is a ".", followed by the internal ID (the Zookeeper instance ID) of the application instance to which requests should be routed, e.g., "abs0000000000").
>
>
>
> When requests reach the load balancer, it first decides which application type is responsible, based on the application's context, e.g., "/abs", "/collaboration" as usual. Then the load balancer checks if a cookie for that application is set and if the cookie is pointing to a currently registered instance. If so, the load balancer sends the request to that instance. If no cookie is set for the application or if the cookie value is pointing to a non-existing or currently not serving instance, the normal round robin approach for request distribution commences.
>
> The cookies are set by the applications themselves: whenever a request reaches an application for which preferred routing is enabled, a servlet filter in that application's filter chain checks if the routing cookie for the application's type is set and that its value is the internal ID of the current application. If this is not the case, the response to the request is enriched with a set-cookie header that sets such a cookie in the client.

8.1.2   Failure handling

If an instance of an application fails, the load balancer will notice this and redirect requests to the remaining instances. Any requests that are handled by the failing instance the moment it crashes will fail, but the clients are usually built to tolerate this situation by retrying the failed request. In the worst case, the users might observe a little hiccup and might have to, e.g., reload the page in the browser. Further, a user who is redirected to a new instance will not benefit from the preferred routing mechanism. The new instance will not yet have the data it is working with in the cache (for that user "the cache is cold"), so the first few requests must again go to the backends and will therefore be a bit slower momentarily.

Such minor glitches or slowdowns in the case of a failure aside, a failed instance will leave you with **reduced capacity** until it becomes available again, but the automatic restart should make sure that this does not take too long. If you want to make sure that even in the face of a failed instance the performance of the system does not suffer too badly, you can do **overprovisioning**, i.e., have more instances of the applications running than is needed for high availability or for the regular system load. For example, assume that your usual system load can be handled with good performance (i.e., acceptable response times) with three instances of an application runnable. While three instances are redundant and allow the system to operate if one or even two of them fail, to avoid degraded performance during such an outage, you might want to add a fourth or fifth instance to your system.

> ℹ️ In the case of an instance failure, the preferred routing approach described in the previous section can be a bit problematic. During the outage of an instance, all clients that were assigned to it will be routed to the remaining instance(s), and will then have the preferred routing cookie set to point to these instances. When the failed instance comes back, the clients will however continue to be routed to the surviving instances, i.e., there is no mechanism in place to redistribute clients to recovered (or to freshly started) instances. The recovered instance will only receive requests from "new" clients that do not yet have a preferred routing cookie. Because the expiry of the preferred routing cookie is set to "session", they will be deleted if the browser is closed, which is a way to reset the preferred routing mechanism for an individual client (or by deleting the browser's cookies).

8.1.3 Distributing stateless applications

As was said above, for most applications, making them highly available is as a simple as configuring additional instances. This works for the business server (ABS), user management (UMCAdmin), document storage (ADSAdmin), Collaboration (ECP), portal (Copernicus), analysis (Octopus), and process governance (APG) components. For the ADS component, you must change the default configuration to enable horizontal scaling and high availability. For the simulation, dashboarding, and ARCM components, the situation is slightly different because these components are either not fully stateless or have other technical restrictions that will be explained below.

8.1.4 Configuring Process Governance for high availability

If you use a multi-node scenario, you can optimize the load balancing of your system. There are two possibilities:

TENANT_BASED (Default)

PROCESS_INSTANCE_BASED

By default, Process Governance follows the tenant-based strategy depicted below. If you have, for example, three nodes you can have three Process Governance runnables installed, one on each server. The number of tenants used can be higher. Let us assume you have five tenants.

When tenants first access Process Governance, they are assigned to the Process Governance instances as follows: Tenant 1 is assigned to Process Governance 1, tenant 2 to Process Governance 2 and so on.
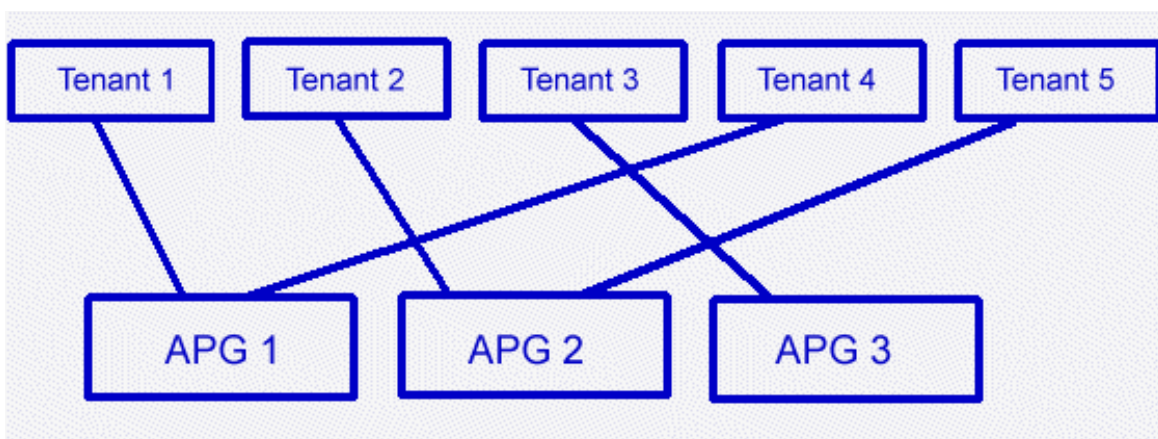
To change this to the process-based strategy, proceed as follows.

You can reconfigure the scalability option of Process Governance using ARIS Cloud Controller (ACC). ACC is a command-line tool for administrating and configuring an ARIS installation. It communicates with ARIS Agents on all nodes. For details refer to ARIS Cloud Controller (ACC) Command-Line Tool.pdf.

1. Start ARIS Cloud Controller (ACC).

2. Stop the Process Governance runnables in all nodes. To do so, enter
   on all nodes stop <Process Governance instance>, for example, **on all nodes stop apg_m**.
   Enter **on all nodes reconfigure<Process Governance instance> "JAVA-Dcom.idsscheer.age.node.assignment.strategy"="PROCESS_INSTANCE_BASED"**, for example, **on all nodes reconfigure apg_m "JAVA-Dcom.idsscheer.age.node.assignment.strategy"="PROCESS_INSTANCE_BASED"**.

3. Start the Process Governance runnables in all nodes. To do so, enter
   on all nodes start <Process Governance instance>, for example, **on all nodes start apg_m**.

You have reconfigured the scalability strategy of Process Governance.
This strategy is depicted below.
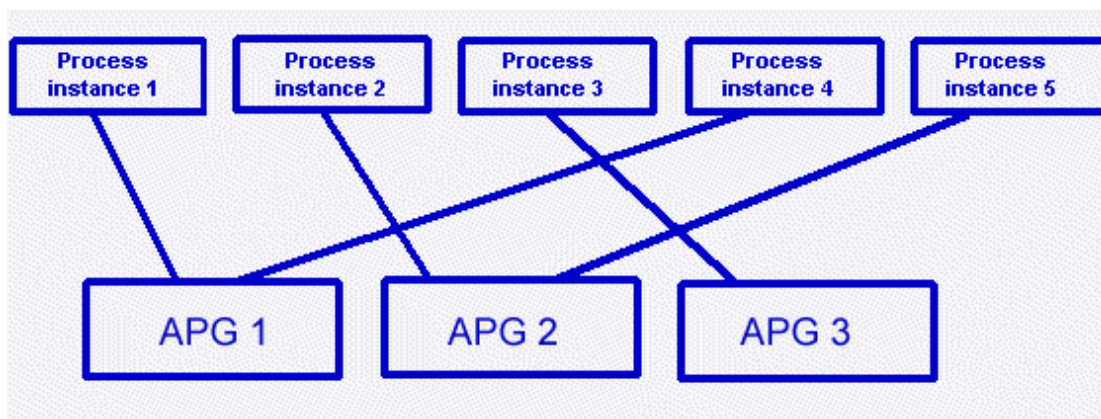
In this scenario, one node can become overloaded, for example, if nodes stop running and the other nodes have to take over.
To switch back to the default behavior, proceed as described above but using the TENANT_BASED parameter.

8.1.5   Rebalance Process Governance nodes

If you use a multi-node scenario, you can optimize the load balancing of your system.
The PROCESS_INSTANCE_BASED assignment strategy is configured for Process Governance.
You can configure the rebalancing of Process Governance nodes using ARIS Cloud Controller
(ACC). ACC is a command-line tool for administrating and configuring an ARIS installation. It
communicates with ARIS Agents on all nodes. For details, please refer to **ARIS Cloud
Controller (ACC)** Command-Line Tool.pdf.

By default, rebalancing is enabled. When you add a new node and activate it, the process
instances from the tenants will be distributed equally to each node automatically.

8.1.6   Disable rebalancing on all nodes

You must do the following for all Process Governance nodes.

1. Start ARIS Cloud Controller (ACC).

2. Stop the Process Governance runnables in all nodes. To do so, enter
    on all nodes stop <Process Governance instance>, for example, **on all nodes
    stop apg_m**.

3. Enter **on <node_name> reconfigure <Process Governance instance> "JAVA-
    Dcom.idsscheer.age.automatic.node.rebalance"="false"**, for example, **on
    node1 reconfigure apg_m "JAVA-
    Dcom.idsscheer.age.automatic.node.rebalance"="false"**.

4. Start the Process Governance runnables in all nodes. To do so, enter
    on all nodes start <Process Governance instance>, for example, **on all nodes
    start apg_m**.

You have disabled the rebalancing of Process Governance.

In this case, when you add a new node and activate it, the process instances are not
rebalanced automatically among all active nodes for any tenant. If you add a new Process
Governance instance, you must specify the same settings for the new Process Governance
instance before you start it.

8.1.6.1   Enable rebalancing on all nodes

1. Start ARIS Cloud Controller (ACC).

2. Stop the Process Governance runnables in all nodes. To do so, enter
    on all nodes stop <Process Governance instance>, for example, **on all nodes
    stop apg_m**.

3. Enter **on all nodes reconfigure <Process Governance instance> "JAVA-
    Dcom.idsscheer.age.automatic.node.rebalance"="true"**, for example, **on all
    nodes reconfigure apg_m "JAVA-
    Dcom.idsscheer.age.automatic.node.rebalance"="true"**.

4. Start the Process Governance runnables in all nodes. To do so, enter
    on all nodes start <Process Governance instance>, for example, **on all nodes
    start apg_m**.

You have enabled the rebalancing of Process Governance.

In this case, when you add a new node and activate it, the process instances from all tenants
will be rebalanced automatically to each node for all tenants.

8.1.6.2   Add a new Process Governance instance

A new Process Governance instance must have the same settings as the whole system has.
By default, rebalancing is enabled. If rebalancing is required to be disabled, the setting of the
new Process Governance instance must be adapted.

1. Start ARIS Cloud Controller (ACC).

2. Add a Process Governance instance. To do so, enter **on <node_name> configure
    <Process Governance instance>**, for example, **on node1 configure apg_m
    apg_m**.

3. To disable rebalancing, enter **on <node_name> reconfigure <Process
    Governance instance> on <server> "JAVA-
    Dcom.idsscheer.age.automatic.node.rebalance"="false"**, for example,
    **on <node_name> reconfigure apg_m on myserver1 "JAVA-
    Dcom.idsscheer.age.automatic.node.rebalance"="false"**.

4. Start the specific Process Governance runnables. To do so enter
    on node <node_name> start <Process Governance instance>, for example, **on
    node1 start apg_m**.

You have disabled the rebalancing of a specific Process Governance instance.

In this case, when you add a new node and activate it, the process instances are either
rebalanced or not depending on the settings.

8.1.6.3   Remove a specific Process Governance instance

You can remove a specific Process Governance instance.

1. Start ARIS Cloud Controller (ACC).

2. Stop the Process Governance instance. To do so, enter
   on <node_name> stop <Process Governance instance> , for example, **on node1 stop apg_m**.

3. Remove the Process Governance instance. To do so, enter **on <node_name> deconfigure<Process Governance instance>**.

You have removed a specific Process Governance instance.

In this case, when you remove a specific Process Governance instance, the process instances are either rebalanced or not depending on the settings.

8.1.7   Configuring document storage for high availability

With ARIS 10, you must configure several settings to make the document storage application highly available and horizontally scalable.

While in ARIS 9 documents were held in the CouchDB runnable which we could make highly available using a master-slave replication mechanism, in ARIS 10 we removed the CouchDB runnable. Document metadata is now stored in the relational database backend. Document content, however, is stored by default in the file system of the respective node in the working directory of the ADSAdmin runnable. If you add multiple ADSAdmin runnables, each will have its own storage, and they will not be able to see each other's documents, which will cause errors.

There are two options to remedy this situation:

1. It is possible to also **store the document content in the database**. To enable this, set the configure parameter "JAVA-Dcom.aris.ads.filesystem.active" to "false" **on all ADSAdmin runnables** in your installation **before** starting any ADSAdmin instance. You can either set this parameter directly when configuring the instances or later using the reconfigure parameter:

    ```
    configure adsadmin_m ads1 JAVA-Dcom.aris.ads.filesystem.active =
    false
    ```

    or

    ```
    reconfigure ads1 JAVA-Dcom.aris.ads.filesystem.active = false
    ```

    respectively.

    ⚠️ If you want to use the database as the backend for document content, then perform the configuration change above before starting to store documents. Otherwise, already existing documents stored in the default location will no longer be accessible.

    The disadvantage of this approach is that storing document content in the DB adds additional load to this backend system. Depending on how intensively you intend to use document storage, this might or might not be a problem. Another problem is that often when you use ARIS with an external database system (Oracle or MS SQL Server), the storage space available for the DBMS (i.e., the size of the tablespace) is rather restricted. So in case you want to use the DBMS as a storage location for document content, you should check with the administrator of your external database system.

2. You can **use a shared network folder** to hold the documents. This shared folder must be made available on each node that has ADSAdmin instances running. The degree of high availability that can be achieved for document storage depends directly on the degree of high availability of the shared network folder.If this is a simple single-node file server, the server will be a single point of failure for the document storage part of ARIS.

The approach to make a shared folder available to ADS differs depending on whether you are using Windows or Linux.

On **Linux** systems, you can mount your shared folder to a local file system location (e.g., /mnt/arisdocs) using the usual operating system mechanism. The details of how to do this are not in the scope of this document. You must make sure that the user under which ARIS is running (by default this is the user "aris", as created by the agent installer package) has permissions to read and write data and create subdirectories in this directory.

You must tell each ADS instance to use the mount point of the shared folder as its document content backend by setting the JAVA-Dcom.aris.ads.filesystem.path configure parameter, either directly when configuring the runnable with configure, e.g., with a command like this:

```
configure adsadmin_<SIZING> <adsInstanceID> JAVA-
Dcom.aris.ads.filesystem.path="<adsDocumentFolderMountPoint>"
```

for example

```
configure adsadmin_m ads JAVA-Dcom.aris.ads.filesystem.path="/mnt/arisdocs"
```

or later with a reconfigure command like this:

```
reconfigure <adsInstanceID> JAVA-
Dcom.aris.ads.filesystem.path="<adsDocumentFolderMountPoint>"
```

for example

```
reconfigure ads JAVA-Dcom.aris.ads.filesystem.path="/mnt/arisdocs"
```

You must do this for all ADS instances in the installation.

⚠️ If you want to change the folder used as the backend for document content, then perform the configuration change above before starting to store documentsOtherwise, already existing documents stored in the default location will no longer be accessible.

On **Windows**, accessing a shared network folder is not as straightforward. By default, the ARIS agent runs with the local SYSTEM user ("LocalSystem" or "NT AUTHORITY\SYSTEM"), which in most environments does not have access to network shares. There are various workarounds for this:

a) You **can change the user under which the ARIS agent is running** to a user who has access to the network share, for example a user in your Windows domain. This approach is **not recommended** and will therefore not be described in detail here.

b) Assuming that the machines on which the ADS runnable instances are running are part of a Windows domain, you can **grant each machine's system user access to the share**. To add a permission for a computer, go to the "Advanced sharing" dialog for the shared folder, then click "Permissions", then "Add", then "Object Types", then select the checkbox "Computer", as illustrated in the figure below.

FIGURE 47
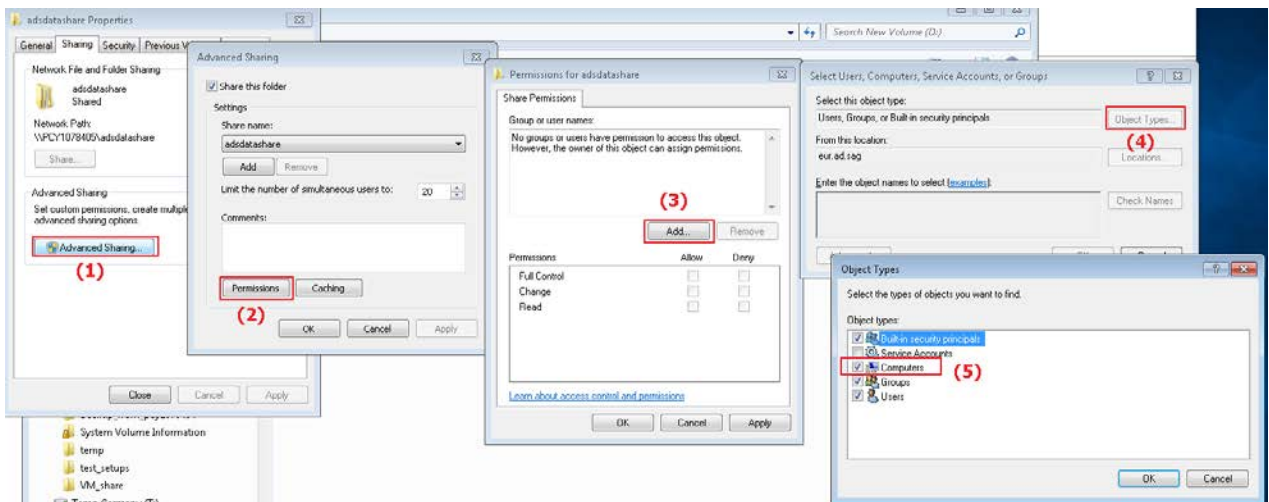
You can then search for the computer name of the machine(s) on which ADSAdmin will be running, add them to the list of permitted users and assign both "Read" and "Change" permissions to each of them, as shown in figure 48.

FIGURE 48
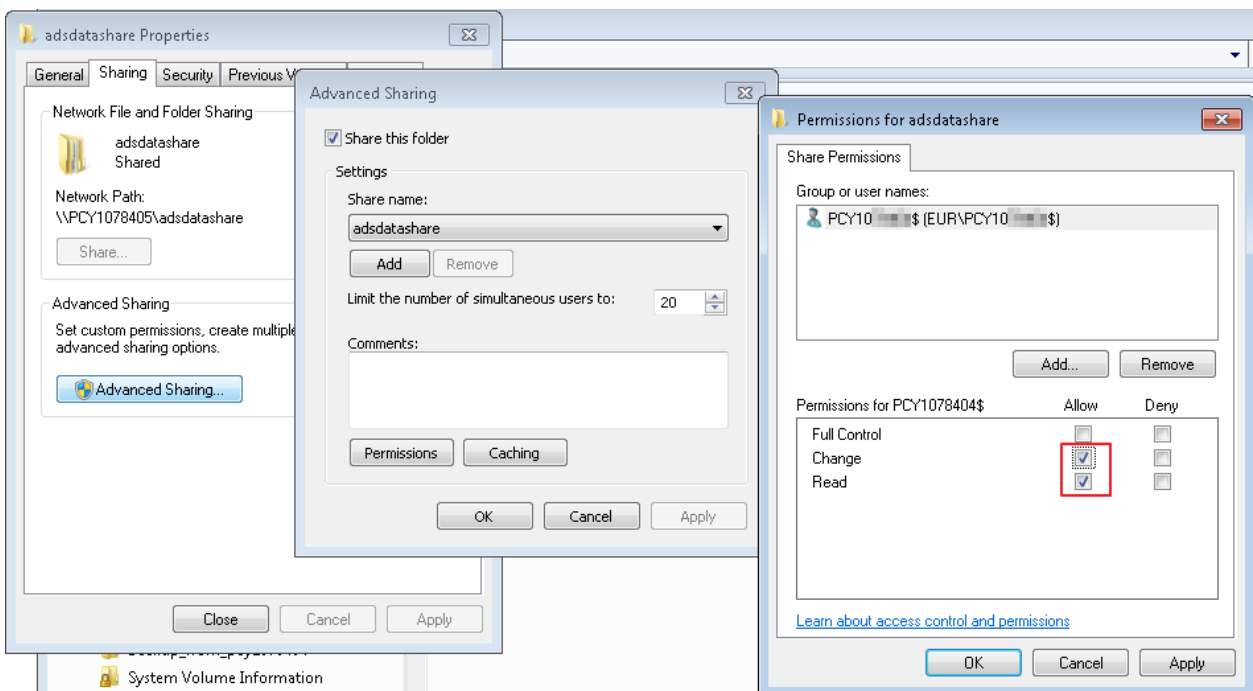
To make ADS use the share, you must set the configure parameter JAVA-Dcom.aris.ads.filesystem.path to point to the share's location, again either directly when configuring the runnable with a command like this

```
configure adsadmin_<SIZING> <adsInstanceID> JAVA-
Dcom.aris.ads.filesystem.path="//<HOSTNAME>/<SHARENAME>"
```

for example:

```
configure adsadmin_m ads JAVA-
Dcom.aris.ads.filesystem.path="//myfileserver.company.com/arisdocuments"
```

or later with a reconfigure command like this

```
reconfigure <adsInstanceID> JAVA-
Dcom.aris.ads.filesystem.path="//<HOSTNAME>/<SHARENAME>"
```

for example

```
reconfigure ads JAVA-
Dcom.aris.ads.filesystem.path="//myfileserver.company.com/arisdocuments"
```

You must do this for all ADS instances in the installation.

ℹ Note that we used forward slashes ("/") in the above commands to specify the path to the shared folder, instead of the Windows standard backslash ("\") character. This is because the backslash character has a special role of an escape character in the ACC grammar and must be escaped by replacing each backslash with four backslashes. Because in Java applications backward and forward slashes can often be used interchangeably in paths, it is much easier to use only forward slashes.

⚠ If you want to change the folder used as the backend for document content, then perform the configuration change above before starting to store documents Otherwise, already existing documents stored in the default location will no longer be accessible.

c) You can also **mount the shared folder as a drive letter in a way that it is available to the system user**. This, however, requires external tools (psexec by Sysinternals) to open a shell in the context of the system user and is therefore not recommended.

8.1.8 Restrictions of the tenant management application (inside UMCAdmin) regarding high availability

The UMCAdmin runnable does not only contain the user management web application, but also the ACC server and tenant management web applications. The ACC server application provides both the healthcheck user interface (which you can access with the URL http://<arisServer>:<arisHttpPort>/acc/ui or https://<arisServer>:<arisHttpsPort>/acc/ui, respectively) and also the create, backup, restore, and delete tenant functionality, on top of which the tenant management web application provides a user interface. The ACC server itself is stateless and can be scaled just like UMC. However, the tenant management application is not fully stateless, as it stores any tenant backups in a local folder (by default the folder "backup" inside the runnable's working directory). Consequently, the backup files are always available only on the instance that created them. So if you have multiple instances of the UMCAdmin runnable (and thus also the tenant management application) in your installation, and if you try to access a backup file, but your request is directed to an instance that does not have this particular file in the local directory, you will encounter an error. There are two solutions to this problem:

Similar to document storage, you can configure the tenant management web application to use a shared network folder. You specify the path to the folder by setting the configure parameter "JAVA-Dtm.backup.folder" directly during the configure command

```
configure umcadmin_m <instanceId> JAVA-Dtm.backup.folder =
"<pathToSharedFolder>"
```

resp.

```
reconfigure umcadmin_m <instanceId> JAVA-Dtm.backup.folder =
"<pathToSharedFolder>"
```

Similar to the use of shared network folders as backend for document storage that we described in the previous section, the shared folder can be given as the name of a network share (e.g., "\\server\share"). Alternatively, a local drive letter to which the network share is mounted can be specified. The same considerations regarding the accessibility of the share for the application as discussed above apply here as well.

Examples

```
configure umcadmin_m umcadmin_m JAVA-Dtm.backup.folder = "n:/tenantBackups"
```

resp.

```
reconfigure umcadmin_m <instanceId> JAVA-Dtm.backup.folder =
"<pathToSharedFolder>"
```

will make tenant management place the backups to the folder "tenantBackups" on drive n, while:

```
reconfigure umcadmin_m JAVA-Dtm.backup.folder =
"//fileserver.company.com/arisTenantBackups"
```

will make TM use the share \\fileserver.company.com/arisTenantBackups.

ℹ Note that just as with ADS we used forward slashes ("/") in the above commands to specify the path to the shared folder instead of the Windows standard backslash ("\") character. This is because the backslash character has a special role of an escape character in the ACC grammar and must be escaped by replacing each backslash with four backslashes. Because in Java applications, backward and forward slashes can often be used interchangeably in paths, it is much easier to use only forward slashes.

⚠ If you want to change the folder used as the storage location for tenant backups, then perform the above configuration change before starting to do backups. Otherwise, already existing backups stored in the default location will no longer be accessible.

While using a network share is the recommended approach, in particular as it will also make sure that your tenant backups are located "outside" the ARIS installation, for small scale scenarios where you do not need the tenant management (and the ACC server) webapps to be highly available, you can also configure **all but one** UMCAdmin instance such that only the UMC web application inside the runnable will be active, by setting the configure parameter "umcOnly" to true, e.g.

```
reconfigure umcadmin m umcOnly=true
```

That way all your backup files will be located on the UMCAdmin instance where this parameter is left unset (resp. is explicitly set to "false").

8.1.9   Restrictions of the simulation component regarding high availability

A simulation always runs in one simulation runnable instance, which holds the current state of the simulation in memory. If the simulation runnable crashes, any simulation runs that were happening on that instance are aborted and must be repeated. Still, it makes sense to have multiple simulation runnables in the system, first to distribute the load of several parallel simulation runs across multiple instances (which should preferably run on different nodes), but it will also allow the users to restart their simulation run immediately instead of having to wait for the crashed instance to be restarted.

8.1.10   Restrictions regarding the Dashboarding ("ARIS Aware") component regarding high availability

Due to technical restrictions, with ARIS 10.0 (and 10.0 SR1), the dashboarding runnable cannot be scaled horizontally as straightforwardly as it is possible with most of the other runnables. While it is possible to have several instances for horizontal scaling for load distribution, you must configure one selected instance to be the master node. The master instance is the only instance that contains the Universal Messaging (UM) component, which is an essential part of the architecture of the underlying MashZone product. The other instances must have UM removed and must be set to act as slaves only. If the master instance fails, dashboarding will be unavailable until it is restarted or until a slave instance has been manually configured to take over the master role.

The procedure to configure multiple dashboarding instances in a master-slave setting differs between ARIS version 10.0.0 (GA) and version 10.0.1 (SR1).

For versions 10.0.2 (SR2) and later, Dashboarding will no longer need special settings to work with multiple instances in a single installation, but can be scaled horizontally without further precautions like most of our other applications.[11]

Configuration for ARIS 10.0 GA

⚠️ Note that most of these configuration steps are not preserved during an update or a reconfigure command and must be re-applied.

To configure one instance of the Dashboarding runnable to be the **master**, perform the following steps:

1. Stop the runnable if it is not already stopped.
2. In the file <runnableWorkingDir>\um\um.properties, add the property "um-server.slaveNodes". The value of this property is the comma-separated list of the hostnames of all slave nodes. For example:
   `um-server.slaveNodes=arisworker1.example.com,arisworker3.example.com`
3. Delete the folder <runnableWorkingDir>\um\presto.
4. Start the runnable.

To configure an instance of the Dashboarding runnable to be a **slave**, perform the following steps:

1. Stop the runnable if it is not already stopped.
2. Delete the entire universal-messaging web application by deleting the folder <runnableWorkinddir>/base/webapps/0um-launcher).
3. In the file <runnableWorkingDir>\runnable.properties, set the value of the property "zookeeper.bundle.parts" only "dashboarding". Find a line that starts with zookeeper.bundle.parts and usually looks like this:
   `zookeeper.bundle.parts=dashboarding,universal-messaging.`
   Change the line by removing the ",universal-messaging" part so that it looks like this:
   `zookeeper.bundle.parts=dashboarding.`
4. In the file <runnableWorkingDir>\um\um.properties, set the value of the property and ensure that the property um-core.sessionFactory.realmUrl.1 points to the hostname and the Universal messaging port (parameter um.port) of the Dashboarding master instance, e.g. nsp://arisworker1:14492
5. Delete the folder <runnableWorkingDir>\um\presto.
6. Start the runnable.

In case of a failover event, you must choose a new master node and configured it manually. You must also adjust the configuration of all remaining slave nodes to point to the new master. Due to the manual effort involved (in particular, you must restore the folder <runnableWorkinddir>/base/webapps/0um-launcher in a former master instance), unless the outage of the node holding the master instance is not expected to be resolved quickly, it is in general more advisable to focus your efforts on bringing the existing master back online.

Configuration for ARIS 10.0 SR1

Master instance

To configure the dashboarding master instance, you must add the configure parameter um-server.slaveNodes or you can set these configure parameters later with a reconfigure command. You set the value of the parameter to a single string (i.e., surrounded by double-quotes) containing the comma-separated list of the full-qualified host names (FQN) and the IP addresses of all slave nodes. For example, if you have two slave instances, the configure parameter takes the form

```
um-server.slaveNodes="<slave1FQN>,<slave2FQN>,<slave1IPAddress>,<slave2IPAddress>"
```

Assuming that your slave nodes are located on the machines arisworker1.example.com and arisworker3.example.com with IP addresses 10.23.42.1 and 10.23.42.3, respectively, the configure parameter is

```
um-server.slaveNodes="arisworker1.example.com,arisworker3.example.com,10.23.42.1,10.23.42.3"
```

Slave instances

To configure a dashboarding instance to act as a slave, you must set two configure parameters. Set "um.deactivated" to "true" to deactivate the Universal Messaging component. Further, set "um-core.sessionFactory.realmUrl.1" to a URL that points to the FQN of the master instance:

```
um-core.sessionFactory.realmUrl.1="nsp://<masterFQN>:${um-server.port}"
```

Assuming that your master instance is located on machine arisworker2.example.com, you set the following two configure parameters on each dashboarding slave instance:

```
um.deactivated = true
um-core.sessionFactory.realmUrl.1="nsp://arisworker2.example.com:${um-server.port}"
```

Configuration for ARIS 10.0 SR2[12]

For versions 10.0.2 (SR2) and later, Dashboarding will no longer need special settings to work with multiple instances in a single installation, but can be scaled horizontally without further precautions like most of our other applications.

---

[12] New content for ARIS 10.0 SR2 release is marked with

8.1.11   Restrictions of the ARCM component regarding high availability

With ARCM, all requests of an active client session are routed to the same ARCM instance using the preferred routing mechanism explained above. However, this is not only used for performance optimization, but it is necessary for correct functioning of the application because the ARCM runnable must hold the current client state in memory. This means that if an ARCM runnable crashes, all clients that were handled by this instance will lose any unsaved data and will be returned to the home screen. As long as other ARCM instances are available, the affected users will be rerouted to one of the remaining instances and can resume their work from the last saved state. Users on other ARCM instances will not be affected. If multiple ARCM runnables are installed, a CDF runnable is mandatory.

8.2   Distributing applications in our example scenario

Continuing with our example scenario from the previous chapters, we will now add redundant application instances for all application types needed for a minimal ARIS Connect installation (i.e., business server (ABS), user management (UMCAdmin), document storage (ADSAdmin), Collaboration (ECP), portal (Copernicus), process governance (APG), Analysis (Octopus), and Simulation.). Figure 49 below shows a possible resulting configuration. Note that this is meant only as an example. Depending on the amount of RAM needed and load observed by the different runnables for your specific usage scenarios, you might notice an imbalance in the memory and CPU usage across these nodes. You can easily adjust for this by deconfiguring the application runnable on one node and configuring it freshly on another node that is currently underused. You should make sure that the two instances of the same application type are always on different nodes.

FIGURE 49: POSSIBLE HIGHLY AVAILABLE DISTRIBUTION OF THE APPLICATION RUNNABLES FOR AN ARIS CONNECT INSTALLATION
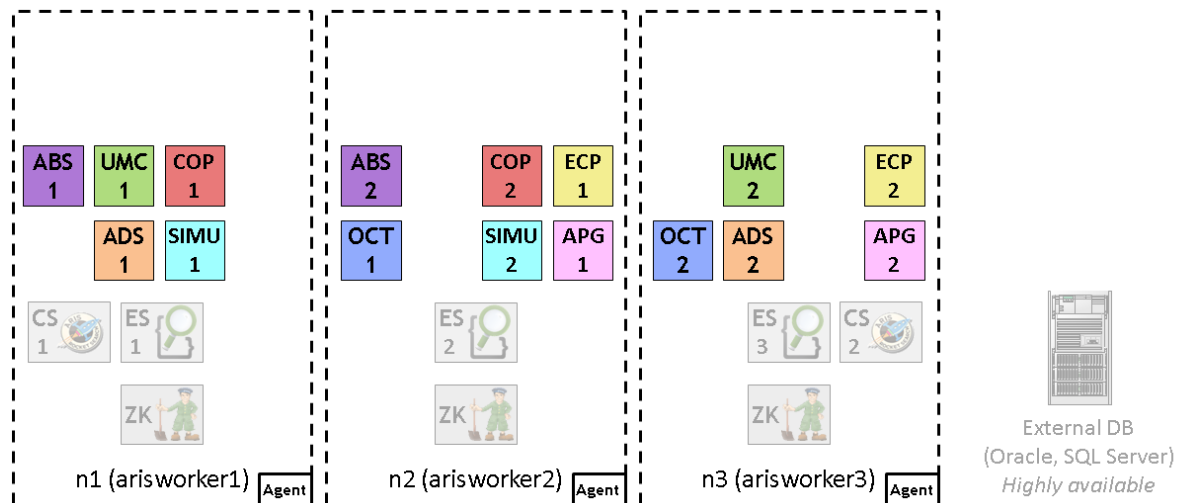
The configure commands required to set up this scenario are the following:

```
on n1 configure abs_m abs1
on n2 configure abs_m abs2

on n1 configure umcadmin_m umc1
on n3 configure umcadmin_m umc2

on n1 configure copernicus_m cop1
on n2 configure copernicus_m cop2

on n2 configure ecp_m ecp1
on n3 configure ecp_m ecp2

on n2 configure octopus_m oct1
on n3 configure octopus_m oct2

on n1 configure adsadmin_m ads1 "JAVA-Dcom.aris.ads.filesystem.active" = "false"
on n3 configure adsadmin_m ads2 "JAVA-Dcom.aris.ads.filesystem.active" = "false"

on n1 configure simulation_m simu1
on n2 configure simulation_m simu2

on n2 configure apg_m apg1
on n3 configure apg_m apg2
```

Observe that the only configure parameters that we had to set were the ones to make ADS use the database as backend for document content storage. All other runnables were left at the default configuration as defined in the "m" sizing of their respective app type.

Depending on how much resources you have available on your three worker nodes, they might or might not be able to hold all these runnables when the system is actively used and the runnables start consuming memory and CPU. In that case you have several options (aside from increasing the amount of hardware resources available):

- If you are interested only in familiarizing yourself with a distributed installation for evaluation purposes and do not plan to use the system for production, you can decide to ignore the resulting performance loss (in particular, if memory runs out, the operating system will start swapping, which will significantly decrease performance).

- Alternatively, you can configure the runnables in small ("s") sizing instead of the medium ("m") sizing we were using. That way, the Java heaps of the individual runnables cannot grow as much. However, certain uses cases might now fail, in particular bulk operations with large amounts of data, running large and complex reports that handle lots of data, etc.. If your interest is focused on one or several applications, such as business server, you can also choose to use "s" sizing for only the less relevant application types and leave the relevant ones at "m". However, be careful **not to have differently sized runnables of the same type** in the system, such as one ABS instance in sizing "m", the other in sizing "s". Having different sizings for different app types is okay, such as all UMC, Octopus, and Simulation runnables configured with "s" sizing, and leaving the others at "m" sizing.

- Another alternative is that you might actively decide to **leave some applications without high availability**. For example, if collaboration or document storage is not essential for your scenario's main use cases, you can configure only a single instance of each and risk an outage of the corresponding part of the application if the single instance crashes or the entire node it resides on goes down. In general, the rest of the application will continue to work fine in such a situation, but (users might have to reload the current portal page.

- Leave out or deactivate some applications entirely. For example, if you are not using, for example, **ARIS Simulation**, you can just leave out those runnables altogether. The application will in general handle this situation gracefully. However, because the functionalities of some applications depend on each other, it is not often easy to decide which ones you can do without. Further, certain UI elements of these applications will remain visible even if they are deactivated both in the thin and rich clients, but using these elements will either not give any feedback at all, yield error messages, or give you empty dialogs. It is therefore not recommended to remove any of the above applications completely from a production system.

- The notable exception are the optional app types that we did not include in the example configuration above, that is the **Heavy-Duty Server** (HDS), the **cdf**, and the **dashboarding** runnables. The **dashboarding** runnable is a server component that provides ARIS Aware dashboards in ARIS Connect. The **hds** runnable is an additional component that reduces the load of ARIS reports running on you **abs** instance. The **cdf** runnable is a queue that can be used to decouple synchronous accesses to services. It is required for communication between the **abs** instance and the **hds** instance. How to use and configure **HDS** is described on page 156.

8.3 Completing the Installation

8.3.1 Adding JDBC drivers

If you are using an external database system (and are not using a single Postgres runnable to keep things simple, albeit not highly available), before you can start your application runnables, you must add the JDBC drivers to each instance. This does not include UMCAdmin, which does not use the DB backend service.

Assuming you used the configure commands as they were given in our example above, and assuming that the JDBC driver JAR was named "driver.jar" and was copied into the subdirectory "jdbc" in your remote repository, you must run the following commands:

Enhancing JDBC drivers (versions <10.0.8)

```
on n1 enhance abs1 with commonsClasspath path "jdbc/driver.jar"
on n2 enhance abs2 with commonsClasspath path "jdbc/driver.jar"

on n1 enhance cop1 with commonsClasspath path "jdbc/driver.jar"
on n2 enhance cop2 with commonsClasspath path "jdbc/driver.jar"

on n2 enhance ecp1 with commonsClasspath path "jdbc/driver.jar"
on n3 enhance ecp2 with commonsClasspath path "jdbc/driver.jar"

on n2 enhance oct1 with commonsClasspath path "jdbc/driver.jar"
on n3 enhance oct2 with commonsClasspath path "jdbc/driver.jar"

on n1 enhance ads1 with commonsClasspath path "jdbc/driver.jar"
on n3 enhance ads2 with commonsClasspath path "jdbc/driver.jar"

on n1 enhance simu1 with commonsClasspath path "jdbc/driver.jar"
on n2 enhance simu2 with commonsClasspath path "jdbc/driver.jar"

on n2 enhance apg1 with commonsClasspath path "jdbc/driver.jar"
on n3 enhance apg2 with commonsClasspath path "jdbc/driver.jar"
```

For versions 10.0.8 or above, you can instead use the "enhanceall" command, which only requires you to specify the app type and will then apply the enhancement to all instances of that app type:

Enhancing JDBC drivers (version >=10.0.8)

```
on all nodes enhanceall abs_m with commonsClasspath path "jdbc/driver.jar"

on all nodes enhanceall copernicus_m with commonsClasspath path
"jdbc/driver.jar"

on all nodes enhanceall ecp_m with commonsClasspath path "jdbc/driver.jar"

on all nodes enhanceall octopus_m with commonsClasspath path
"jdbc/driver.jar"

on all nodes enhanceall adsadmin_m with commonsClasspath path
"jdbc/driver.jar"

on all nodes enhanceall simulation_m with commonsClasspath path
"jdbc/driver.jar"

on all nodes enhanceall apg_m with commonsClasspath path "jdbc/driver.jar"
```

ⓘ About the semantics of the "enhanceall" command

Note that while we have different app types in the generated.apptypes.cfg for a certain application to represent different "sizings" (e.g., "abs_s", "abs_m", "abs_l", …), for the purpose of the enhanceall command, all app types for one specific application are effectively equal (as they all describe the same application artifact and only differ in some of their parameters). So in the example above, instead of using the "m" sizing that we also used when we configured the runnables (and where the sizing *does* make a difference), here we could also have used the "s" or "l" sizing instead without changing the outcome of the command.

8.3.2 Adding help files

By default, the application runnables do not contain the files of the online help. You must add these files using the help enhancement. You specify the files by using the "artifact coordinates", that is the technical name and version of the file.

The basic structure of an enhance command for adding the help for most applications is

```
on <nodeName> enhance <instanceId> with help <helpArtifact> <version> type war
```

where
<nodeName> is the logical name of the node, e.g. n1
<instanceId> is the instance ID of the runnable to which you want to add the help, e.g, "abs1"
<helpArtifact> is the technical name of the actual set of help files, e.g. com.aris.documentation.architect. The technical name is different for the help of different runnables.
<version> is the version you are installing, e.g., "10.0.0".
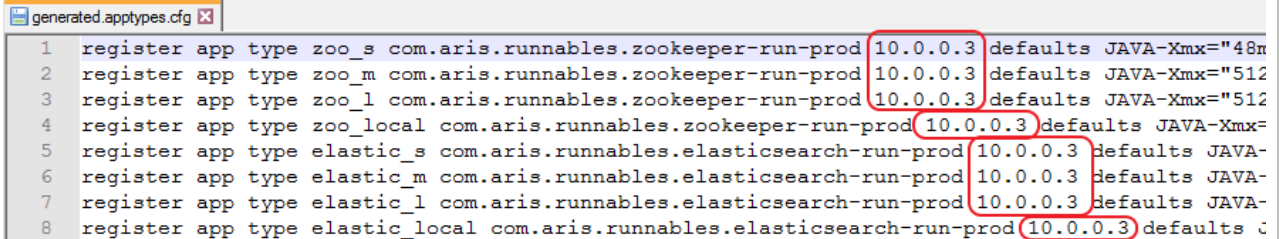
> ℹ To find out the actual version of the runnables and help files on your DVD, check the generated.apptypes.cfg file. It contains multiple "register app type" commands, which contain the versions of the runnables - the versions used here are also the versions of the help files. For example, if your DVD contains ARIS version 10.0.0.3, the generated.apptypes.cfg file looks like this:

For the business server runnable (ABS), some parts of the online help have been separated into dedicated files for different languages. The language can be specified with an additional option, the "classifier", in the enhance command. So for adding help to ABS, the structure of the enhance command is

```
on <nodeName> enhance <instanceId> with help <helpArtifact> <version>
classifier <lang> type war
```

The variables <nodeName>, <instanceId>, <helpArtifact>, and <version> have the same meaning as above.
<lang> is the language of the help artifact. Currently, there are ABS help files for five different languages: en (English), de (German), fr (French), es (Spanish), ja (Japanese). You can add all languages using several enhance commands, or you can pick only those languages that you need, depending on your user base. Adding the English language version (en) is mandatory.

The names and classifiers of the help artifacts of the different runnables are the following:

Runnable	Help artifacts (since version)	Languages
abs	com.aris.documentation.architect	de, en, fr, es, ja pt (since 9.8.2) ru, zh, nl (since 9.8.6)
	com.aris.documentation.architect-scripthelp	-
	com.aris.documentation.architect-methodhelp	-[13], de, en, fr, es, ja pt, ru, zh, nl (since 9.8.6)
adsadmin	com.aris.documentation.administration.ads	-
umcadmin	com.aris.documentation.administration.administration com.aris.documentation.administration.tm com.aris.documentation.administration.infrastructure (since 10.9.0)	- -
copernicus	com.aris.documentation.connect	-
apg	com.aris.documentation.processboard	-
businesspublisher	com.aris.documentation.publisher.publisher	-
arcm	com.aris.documentation.arcm.arcm	-

Again assuming that you are following our example exactly, and assuming that you are using repository of version 10.0.0, the commands to add all available help files to your application runnables are the following:

---

[13] The methodhelp consists of a common part (an artifact without language classifier) and parts for the different languages. The common part should always be enhanced, while the language specific parts can be chosen depending on the languages required by one's user base.

Help enhancements for the ABS runnables

```
on n1 enhance abs1 with help com.aris.documentation.architect 10.0.0 classifier de type war
on n1 enhance abs1 with help com.aris.documentation.architect 10.0.0 classifier en type war
on n1 enhance abs1 with help com.aris.documentation.architect 10.0.0 classifier fr type war
on n1 enhance abs1 with help com.aris.documentation.architect 10.0.0 classifier es type war
on n1 enhance abs1 with help com.aris.documentation.architect 10.0.0 classifier ja type war
on n1 enhance abs1 with help com.aris.documentation.architect 10.0.0 classifier pt type war
on n1 enhance abs1 with help com.aris.documentation.architect 10.0.0 classifier ru type war
on n1 enhance abs1 with help com.aris.documentation.architect 10.0.0 classifier zh type war
on n1 enhance abs1 with help com.aris.documentation.architect 10.0.0 classifier nl type war
on n1 enhance abs1 with help com.aris.documentation.architect-scripthelp 10.0.0 type war
on n1 enhance abs1 with help com.aris.documentation.architect-methodhelp 10.0.0 type war
on n1 enhance abs1 with help com.aris.documentation.architect-methodhelp 10.0.0 classifier de type
war
on n1 enhance abs1 with help com.aris.documentation.architect-methodhelp 10.0.0 classifier en type
war
on n1 enhance abs1 with help com.aris.documentation.architect-methodhelp 10.0.0 classifier es type
war
on n1 enhance abs1 with help com.aris.documentation.architect-methodhelp
10.0.0 classifier fr type war
on n1 enhance abs1 with help com.aris.documentation.architect-methodhelp 10.0.0 classifier ja type
war
on n1 enhance abs1 with help com.aris.documentation.architect-methodhelp 10.0.0 classifier pt type
war
on n1 enhance abs1 with help com.aris.documentation.architect-methodhelp 10.0.0 classifier ru type
war
on n1 enhance abs1 with help com.aris.documentation.architect-methodhelp 10.0.0 classifier zh type
war
on n1 enhance abs1 with help com.aris.documentation.architect-methodhelp 10.0.0 classifier nl type
war


on n2 enhance abs2 with help com.aris.documentation.architect 10.0.0 classifier de type war
on n2 enhance abs2 with help com.aris.documentation.architect 10.0.0 classifier en type war
on n2 enhance abs2 with help com.aris.documentation.architect 10.0.0 classifier fr type war
on n2 enhance abs2 with help com.aris.documentation.architect 10.0.0 classifier es type war
on n2 enhance abs2 with help com.aris.documentation.architect 10.0.0 classifier ja type war
on n2 enhance abs2 with help com.aris.documentation.architect 10.0.0 classifier pt type war
on n2 enhance abs2 with help com.aris.documentation.architect 10.0.0 classifier ru type war
on n2 enhance abs2 with help com.aris.documentation.architect 10.0.0 classifier zh type war
on n2 enhance abs2 with help com.aris.documentation.architect 10.0.0 classifier nl type war
on n2 enhance abs2 with help com.aris.documentation.architect-scripthelp 10.0.0 type war
on n2 enhance abs2 with help com.aris.documentation.architect-methodhelp 10.0.0 type war
on n2 enhance abs2 with help com.aris.documentation.architect-methodhelp 10.0.0 classifier de type
war
on n2 enhance abs2 with help com.aris.documentation.architect-methodhelp 10.0.0 classifier en type
war
on n2 enhance abs2 with help com.aris.documentation.architect-methodhelp 10.0.0 classifier es type
war
on n2 enhance abs2 with help com.aris.documentation.architect-methodhelp 10.0.0 classifier fr type
war
on n2 enhance abs2 with help com.aris.documentation.architect-methodhelp 10.0.0 classifier ja type
war
on n2 enhance abs2 with help com.aris.documentation.architect-methodhelp 10.0.0 classifier pt type
war
on n2 enhance abs2 with help com.aris.documentation.architect-methodhelp 10.0.0 classifier ru type
war
on n2 enhance abs2 with help com.aris.documentation.architect-methodhelp 10.0.0 classifier zh type
war
on n2 enhance abs2 with help com.aris.documentation.architect-methodhelp
10.0.0 classifier nl type war
```

Help enhancements for the UMCAdmin runnables

```
on n1 enhance umc1 with help com.aris.documentation.administration.administration 10.0.0 type war
on n1 enhance umc1 with help com.aris.documentation.administration.tm 10.0.0 type war


on n3 enhance umc2 with help com.aris.documentation.administration.administration 10.0.0 type war
on n3 enhance umc2 with help com.aris.documentation.administration.tm 10.0.0 type war
```

Help enhancements for the Copernicus runnables

```
on n1 enhance cop1 with help com.aris.documentation.connect 10.0.0 type war
on n2 enhance cop2 with help com.aris.documentation.connect 10.0.0 type war
```

Help enhancements for the ADSAdmin runnables

```
on n1 enhance ads1 with help com.aris.documentation.administration.ads 10.0.0 type war
on n3 enhance ads2 with help com.aris.documentation.administration.ads 10.0.0 type war
```

Help enhancements for the APG runnables

```
on n2 enhance apg1 with help com.aris.documentation.processboardhelp 10.0.0 type war
on n3 enhance apg2 with help com.aris.documentation.processboardhelp 10.0.0 type war
```

9 Highly available client access to a distributed installation

9.1 The roles of the ARIS load balancer (ARIS LB)

In our chapter about the distribution of micro-service applications, we already briefly explained the role of the load balancer both as a single point of access for all kinds of clients and its role in the preferred routing mechanism that we use to improve performance. However, we did not explain the rationale behind routing all clients' requests through the load balancer. In particular, in a single-node installation without redundant applications, it might seem that because there is no load distribution to be done among different instances of the same application type, we could do entirely without a load balancer. However, there is an important restriction that makes it necessary to use a load balancer also in this case.

9.1.1 The same origin policy

The ARIS connect portal is based on a framework for the creation of modern interactive web applications that contain content and scripts that can come from different applications, which are then integrated to offer a single, seamless user experience. However, the same origin policy, an essential security model of all browser-based applications, allows scripts in a page to execute only if they are loaded from the same origin as the main page, where "origin" means the combination of URI scheme (i.e., "http" or "https"), hostname (or IP address), **and port**.

So only if from the perspective of the browser all scripts seem to be coming from the same origin, can the application function properly. If we do not use a loadbalancer and load content and scripts directly from the different applications via their respective HTTP connector port, the same origin policy does not allow them to be executed (as even if the scheme and port are the same, the ports, and thus the "origin" are different).

If, however, all applications are accessed through the load balancer, from the perspective of the browser it will appear as if all scripts are coming from the same server (same scheme, host, **and** port), thus the same origin policy is fulfilled and the scripts can be executed to offer the connect portal functionality.

9.1.2 TLS termination proxy

Another role of the load balancer is that of handling decryption and encryption of the traffic between the clients and the server installation, i.e., serving as a TLS termination proxy. Instead of having full end-to-end encryption of client requests and responses from the client down to the individual application servers and back, the ARIS security model assumes that communication among the runnables is happening in a secure environment and thus need not be encrypted. This not only makes it much easier to use encrypted communication, as you must provide only an SSL certificate for the load balancer(s), but it also takes the load of encrypting and decrypting from the applications and puts it to the load balancers, which are much better suited to the tasks and can be scaled independently.

9.1.3   Load balancing and high availability

Aside from allowing us to work around the same origin policy and handling encryption, in a distributed, highly available installation, the load balancer must fulfill its main role - i.e., balancing the load of all client requests more or less equally across all instances of our applications.

However, if all requests go through the load balancer, this makes it a single point of failure. For this reason, you can have multiple load balancers in a distributed installation, and each one will allow access to the complete application. If, for example, we add a load balancer runnable to each of our three nodes of our example scenario, clients can access the application via the hostnames or IP addresses of any of the worker nodes.

This approach, while possibly acceptable for small scale scenarios, has several limitations. First, the users must be made aware of the fact that they can access the same application via multiple hostnames, which can be confusing for them. More importantly, if one of the load balancers (or the node it is running on) fails, all users that chose this machine for accessing ARIS will lose the connection. Most ARIS clients can deal with connection losses. Unsaved changes are stored on client side. If the outage persists, the users should manually switch over to one of the remaining nodes.

There are two solutions to this problem: one is the use of a round-robin DNS. Here a single hostname is chosen for the ARIS application. In the DNS server, the IP addresses of all ARIS workers with load balancers running on them are registered. Clients will use the single hostname to connect to the application, upon which the DNS will return an arbitrary worker node address.

Round-robin DNS has several drawbacks. While in theory it allows you to hide the fact that the application is distributed to the users to some degree, it will not help in the case of an outage, as clients that were connected to the failed load balancer instance will lose the connection. In addition, a vanilla DNS server does not know which of the servers are currently available and will return the IP addresses of failed instances. Further, the caching of DNS lookups on the client side is hard to influence. Even if the DNS server would check the workers for availability and take unavailable ones out of the rotation, clients could still try to access the unavailable instances. Since in practice these issues are hard to overcome, using round-robin DNS is not recommended.

A more suitable approach is to put a highly available load balancer (HA LB) in front of the ARIS LBs. Such an HA LB is usually a hardware appliance (like "F5") or specialized software on highly available hardware and serves as a single point of entry for their users. The details of how to configure such a device are out of the scope of this manual. All worker nodes holding ARIS LBs must be added as workers to the HA LB's configuration.

Usually, the HA LB also takes over the role of the TLS termination proxy, i.e., it decrypts the requests from clients that are communicating with the ARIS server in encrypted form via TLS, then forwards the decrypted requests to one of the ARIS LBs, receives the unencrypted response from the application via the ARIS LB, then encrypts this response and sends it back to the clients. If the HA LB takes over this responsibility, you do not need to worry about

adding SSL certificates to the ARIS LBs, you can even disable HTTPS entirely on them by setting the configure parameter HTTPD.ssl.port to "0".

9.1.4   The X-Forwarded-* headers

When you use an HA LB in front of the ARIS LBs, you should consider the correct handling of the different **X-Forwarded-*** headers, in particular X-Forwarded-For.

The **X-Forwarded-For (XFF) HTTP** header is a de-facto standard method for identifying the originating IP address of a client accessing a web application (like ARIS) through an HTTP proxy or load balancer.

To understand how this header works, consider the situation shown in Figure 50:

FIGURE 50: WITHOUT X-FORWARDED-FOR, APPLICATIONS SEE THE HA LB'S IP ADDRESS AS APPARENT CLIENT IP ADDRESS
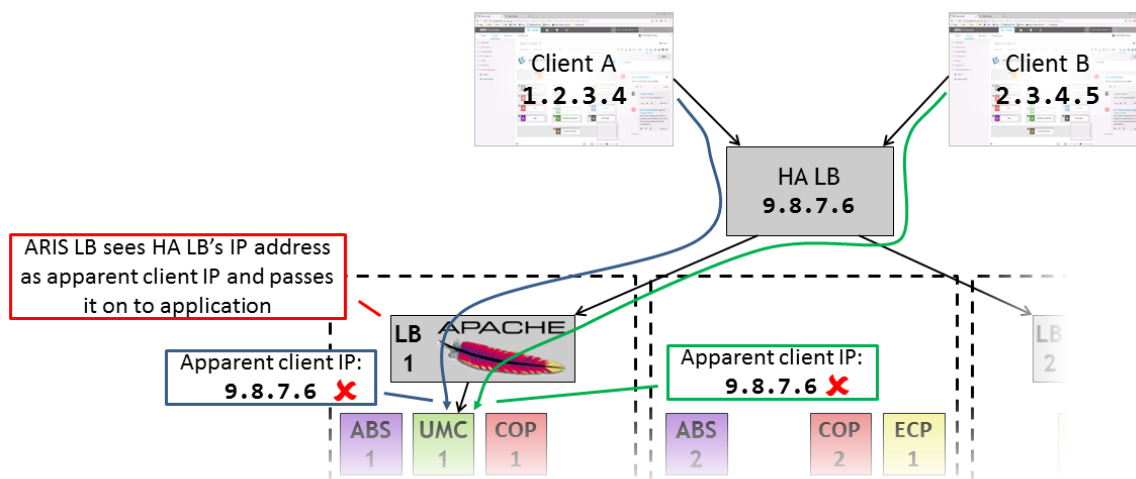
Here, two clients are accessing ARIS through an HA LB. From the perspective of the ARIS LBs, the request is coming from the IP address **9.8.7.6** of the HA LB, and this IP address is passed on to the applications as an alleged client IP address. The result is that from the perspective of the application, all requests seem to come from a single client IP address. Amongst other negative effects, this makes troubleshooting more difficult.

When the HA LB is configured to send an **X-Forwarded-For** header whose value is the actual client IP address with each request, the ARIS LB can pass this information on to the applications, resolving the problem, as shown in Figure 51:

FIGURE 51: HA LB SENDS X-FORWARDED-FOR, APPLICATIONS SEE ACTUAL CLIENT IP ADDRESS
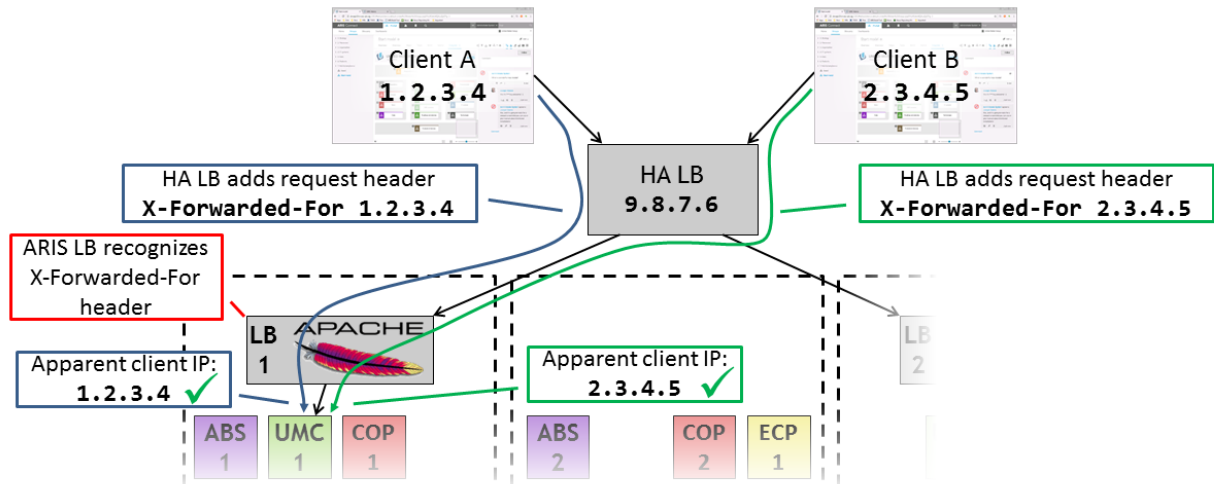
If the ARIS LB simply accepted **X-Forwarded-For** headers from any source, this would allow an attacker to fake its IP address (as it is seen by the ARIS applications) by itself adding a faux **X-Forwarded-For** header. UMC has an (optional) security mechanism in place to prevent attackers from performing arbitrarily many login attempts to "try out" or "guess" a user's password (a form of brute-force attack): if too many failed login attempts for a certain user account coming from a certain IP address are observed, that combination of user account and IP address is locked for any further login attempts for a configurable amount of time.

Up to ARIS version 10.0.1, the ARIS LB would accept **XFF** headers in any request. As a consequence, an attacker could set this header to make ARIS believe that his requests are coming from a different than his actual IP address. That way, the protection mechanism against brute force login attacks could be circumvented, allowing the attacker to try out arbitrary many passwords using different "fake" IP addresses in the **XFF** header, a situation is shown in Figure 52:

FIGURE 52: ATTACKER ABUSING X-FORWARDED-FOR TO FAKE IP ADDRESSES
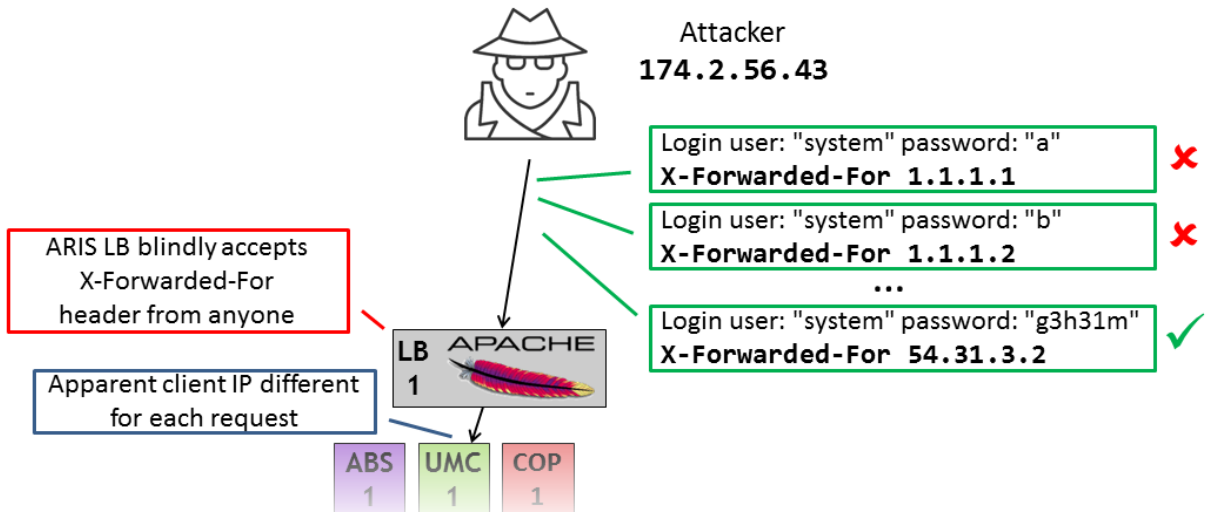
To prevent this problem, starting with ARIS 10.0.2, the ARIS LB by default discards any **XFF** headers received from clients except requests coming from localhost (127.0.0.1), as shown in Figure 53:

FIGURE 53: ARIS LB IGNORES X-FORWARDED-FOR HEADERS BY DEFAULT, PREVENTING BRUTE-FORCE PASSWORD ATTACKS
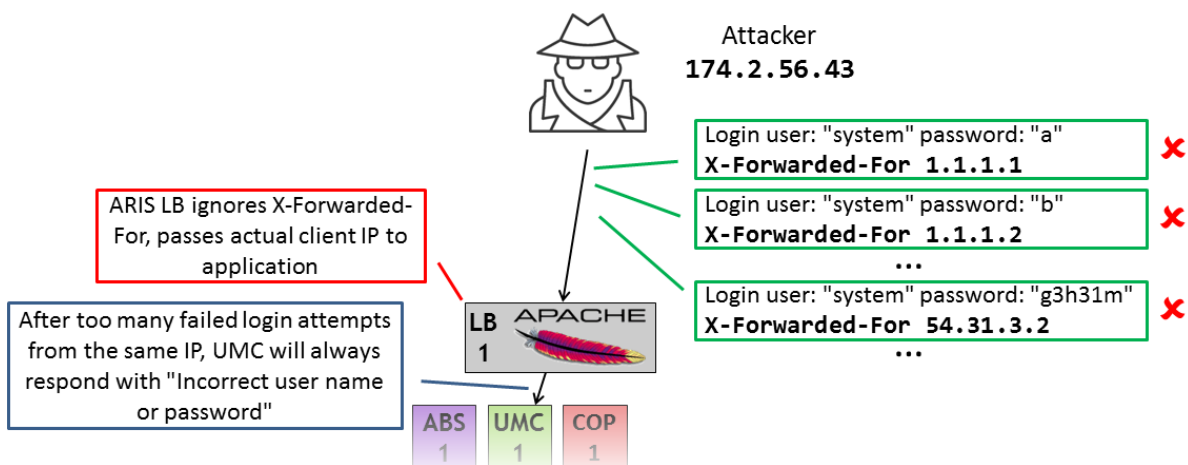
Since the applications (in particular UMC) now see the actual client IP, the attack prevention mechanism works as intended: after too many failed login attempts coming from the same IP

address, UMC always responds with "Incorrect user name or password", even if the attacker did find the correct username/password combination.

While this change will foil such brute-force password guessing attacks (in particular in simple setups, where the ARIS LB is directly exposed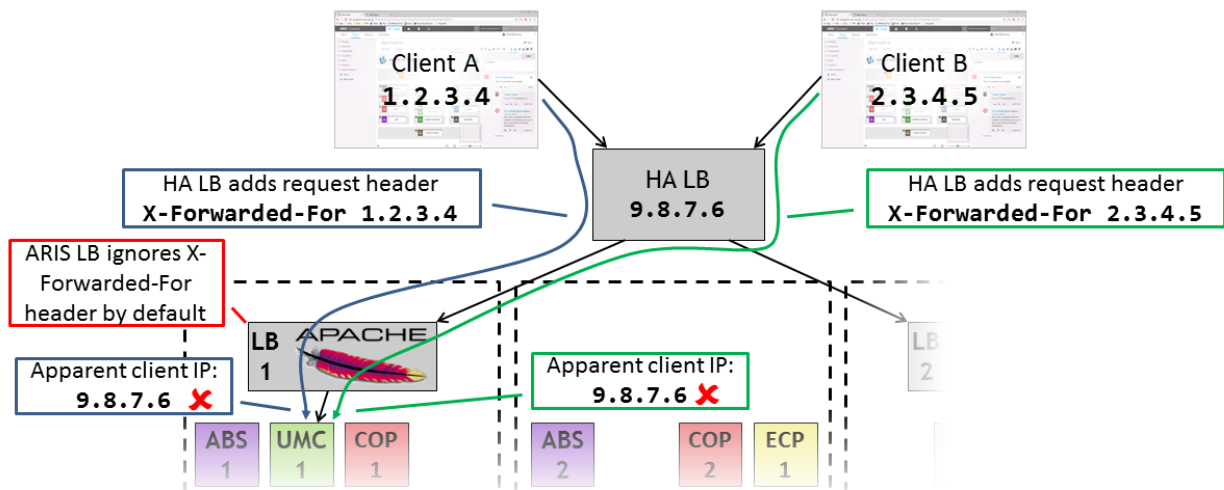 to clients, and thus also to possible attackers), it disables the **X-Forwarded-For** mechanism also for the legitimate use case of putting ARIS behind an HA LB, as illustrated in Figure 54:

FIGURE 54: PROBLEM WITH IGNORED X-FORWARDED-FOR HEADERS IN HA LB SCENARIO

For these scenarios, it is possible to configure the IP address(es) from which the ARIS LB will accept **XFF** headers. To do this, you can use the **HTTPD.X-Forwarded-For.trusted.proxy.regex** load balancer configure parameter, with which you can provide a regular expression (regex, see https://httpd.apache.org/docs/2.4/glossary.html#regex) that will make the ARIS LB accept **XFF** headers from any IP address that matches this regex. Note that any backslash characters that appear in your regex must be escaped by replacing them with four backslash characters. For example, to accept **XFF** headers from the IP address **9.8.7.6** of the HA LB in the example above, you set the **HTTPD.X-Forwarded-For.trusted.proxy.regex** to **"9\\\\.8\\\\.7\\\\.6"**. Observe the use of the quadruple backslashes to escape the dot (".") character that has a special meaning in regular expressions. The resulting scenario is shown in Figure 55:

FIGURE 55: ARIS LB ACCEPTS X-FORWARDED-FOR HEADERS ONLY FROM THE HA LB
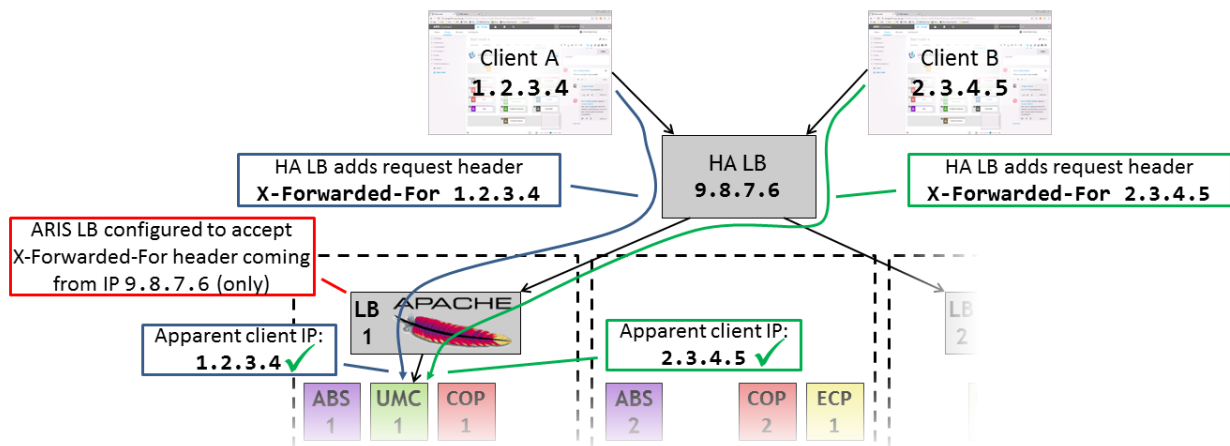
To obtain the behavior as it was in previous versions (i.e., the scenario shown in Figure 51, where the ARIS LB would accept the **XFF** header from any client), you can set the **HTTPD.X-Forwarded-For.trusted.proxy.regex** parameter to **".*"** (a regular expression matching every IP address). This is an acceptable setting if you ensure that clients (and thus possible attackers) cannot access the ARIS LBs directly and that your HA LB itself properly handles **XFF** headers coming from clients (i.e., in general by discarding them).

9.2 Configuring a load balancer in a distributed, highly available installation

Now that we learned about the roles of the ARIS LBs, we need to make sure that all LBs are configured correctly. First, you need more than one ARIS LB in your installation. Further, when following the recommended approach of putting an HA LB in front of your ARIS LBs, you must make some configuration changes.

9.2.1 Make the ARIS LBs work with an HA loadbalancer

To make the ARIS LBs work together with an HA LB, make the following changes in their configuration:

1) Each ARIS LB must register itself in Zookeeper with the scheme, host, and port of the HA LB instead of using the default (The default is to use the hostname or IP of the machine on which the ARIS LB is running, the ARIS LB's HTTPS port and the scheme "https" if HTTPS is enabled on the ARIS LB; or the HTTP port and scheme "http" if HTTPS is disabled). To make this happen, you must set several configure parameters properly on *all* ARIS LBs in the installation. First, you must set

- the parameter "HTTPD.servername" to the hostname or IP address of the HA LB

If you want to make the URLs sent out by the application use https (which requires that you have https enabled on your HA LB), you must set

- the parameter "HTTPD.zookeeper.application.instance.port" to the port on which the HA LB is receiving HTTPS requests (by default that is usually 443) and set
- the parameter "zookeeper.application.instance.scheme" to "https" Otherwise, If you prefer the URLs to point to the unencrypted HTTP endpoint of the HA LB (or if you do not support HTTPS on the HA LB), you must set
- the parameter "HTTPD.zookeeper.application.instance.port" to the port on which the HA LB is receiving HTTP requests (by default that is usually 80) and set
- the parameter "zookeeper.application.instance.scheme" to "http"

The Zookeeper registration of your load balancers is needed because applications that send links to the application (in particular links generated into notification emails) must know through which host the users access the applications, so that they can use this machine's host and the proper scheme and port of the load balancer when building the links. For this, the applications do a lookup of the load balancer's information in Zookeeper. By setting the above configuration parameters, you make your ARIS LBs "fake" their entries, so that any generated links point to the HA LB instead of pointing directly to any of the ARIS LBs.

2) You must disable the FQN redirect by setting the configure parameter "HTTPD.EnforceFQN" to "false".

3) You must make sure that each ARIS LB accepts the **X-Forwarded-For** header that is sent by the HA LB in front of ARIS. For that, you must know the IP address of the HA LB which will be used for communication with the ARIS LBs. You then must create a regular expression

that matches only this IP address and escape it as described in the section about the handling of the **X-Forwarded-For** header above, by replacing each backslash character in your regular expression by four backslashes. For example, if the HA LB has the IP address 9.8.7.6 and you want to make sure that the X-Forwarded-For header is only ever accepted from this IP address by the ARIS LBs, you set the ARIS LB's configure parameter **HTTPD.X-Forwarded-For.trusted.proxy.regex** to "9\\\\.8\\\\.7\\\\.6".

To make 1), 2), and 3) happen, use the following configure parameters command on each of your ARIS LBs (either directly when configuring them, or afterwards, within a reconfigure command):

```
HTTPD.servername=<hostnameOfFakeHaLb>
HTTPD.zookeeper.application.instance.port=<httpPortOfHaLb>
zookeeper.application.instance.scheme="http" HTTPD.X-Forwarded-
For.trusted.proxy.regex=<escapedRegexMatchingIpOfHaLb>
```

if you want to use HTTP, or, if you want to use HTTPS, to

```
HTTPD.servername=<hostnameOfFakeHaLb>
HTTPD.zookeeper.application.instance.port=<httpsPortOfHaLb>
zookeeper.application.instance.scheme="https" HTTPD.X-Forwarded-
For.trusted.proxy.regex=<escapedRegexMatchingIpOfHaLb>
```

if you want to use HTTPS, respectively.

9.3 The load balancers in our example scenario

Returning to our example scenario, we will now add two load balancers to our nodes n1 and n3 and configure them as described above so that they work properly with a highly available load balancer (HA LB), as shown in figure 56.

FIGURE 56: REDUNDANT LOADBALANCERS IN OUR EXAMPLE SCENARIO
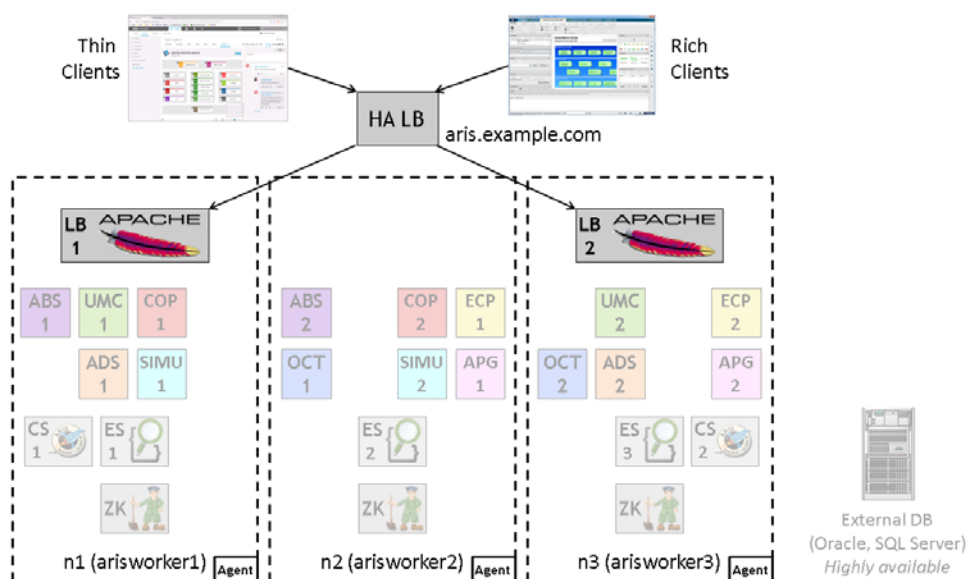
Assuming that the HA load balancer can be reached by the clients through the hostname "aris.company.com", further assuming that it has the IP address **9.8.7.6**, and assuming that

we want any generated links that point to this machine are using HTTPS protocol (i.e., communication is encrypted with SSL/TLS) and the HTTPS standard port, we can configure the load balancers with the following commands (adjust the value given for HTTPD.zookeeper.application.instance.port in case you are using a non-standard port for HTTPS on your HA load balancer):

Configuring the ARIS loadbalancers to work with a HA LB with HTTPS support

```
on n1 configure loadbalancer_m lb1 HTTPD.servername="aris.example.com" \
HTTPD.zookeeper.application.instance.port=443 \
zookeeper.application.instance.scheme="https"

on n3 configure loadbalancer_m lb2 HTTPD.servername="aris.example.com" \
HTTPD.zookeeper.application.instance.port=443 \
zookeeper.application.instance.scheme="https"
```

If you prefer that any links generated by the application use HTTP protocol or in case you do not have HTTPS configured on your HA load balancer, you should configure your ARIS LBs with the following commands (adjust the value given for HTTPD.zookeeper.application.instance.port in case you are using a non-standard port for HTTP):

Configuring the ARIS loadbalancers to work with a HA LB without HTTPS support

```
on n1 configure loadbalancer_m lb1 HTTPD.servername="aris.example.com" \
HTTPD.zookeeper.application.instance.port=80 \
zookeeper.application.instance.scheme="http"
on n3 configure loadbalancer_m lb2 HTTPD.servername="aris.example.com" \
HTTPD.zookeeper.application.instance.port=80 \
zookeeper.application.instance.scheme="http"
```

If you do not have access to an HA loadbalancer, the next section describes how to simulate such a device with software (purely for educational purposes).

In case you do not want to use an HA LB at all (neither a real one nor the "fake" software solution), you should instead leave the ARIS LBs at their default configuration, i.e., instead of the configure commands above, you should use the following commands:

Configuring the ARIS loadbalancers wo work without a HA LB

```
on n1 configure loadbalancer_m lb1

on n3 configure loadbalancer_m lb2
```

9.4 Optional - Simulating a highly available hardware load balancer with software (for educational purposes only)

In case you want to experiment with the behavior and configuration of ARIS when it is put behind a highly-available (HA) load balancer (LB), but do not have access to such a device, this section will provide a description of how to set up a "fake" HA LB by using software (by installing an Apache HTTPD that is configured to act as a reverse proxy in front of the two ARIS LBs in our example scenario). This will give you a single point of entry for clients to the application that should behave like na HA LB, but **will not be truly highly available** and therefore **must not be used in production environments**, but purely for testing and educational purposes. Consequently, we will also not worry about how to secure the Apache installation, so **do not put this on a machine that is publicly accessible on the internet**. Further, we will also not discuss how to enable support for HTTPS on your "fake" HA LB. Therefore, for this example to work, you must use the configure commands for the ARIS LBs found in the section "Configuring the ARIS LBs to work with an HA LB without HTTPS support" in the previous section.

> ⓘ If you are interested in HTTPS support for your "fake" HA LB, you can follow one of the many available tutorials about this topic, for example the one provided by the Apache documentation[14]. In this case you should add the configuration settings we describe below to the VirtualHost section for the HTTPS endpoint.

Throughout this section, we will assume that in addition to the three worker machines (arisworker1, arisworker2, arisworker3) that we used so far, you have another machine available on which you can install our "fake" HA LB. We will assume that this machine's hostname is "aris.company.com". You should replace all hostnames with those in your environment. Further, the descriptions below are for installing Apache on a Windows machine, but the changes to the actual Apache configuration file itself will be the same regardless of the platform it is running on. If you do not have another machine available for running the "fake" HA LB, you can also install it on one of your three worker machines (or directly on your workstation).

---

[14]Apache SSL/TLS Strong Encryption: How-To - https://httpd.apache.org/docs/2.4/ssl/ssl_howto.html

9.4.1   Setting up Apache HTTPD

The first step is to install an Apache HTTPD, which will act as our simulated "hardware" load balancer on the "myaris" machine.

You can use any recent 2.4.x version of Apache HTTPD for Windows.
Install Apache into a directory of your choice. We will refer to this directory as <apacheInstallDir>.

After installing, make sure that Apache itself works by running the httpd.exe located in <apacheInstallDir>\bin from a shell and accessing the aris.company.com machine from a web browser (usually, the default config will configure Apache to run on port 80, so "localhost" will do.

You should see the familiar "It works!" message:

FIGURE 57

Now you must modify the httpd.conf file, which you will find in the <apacheInstallDir>\conf directory.

In the initial section of this file where the "LoadModule" statements are found, make sure that the lines for the modules

mod_proxy, mod_proxy_balancer, mod_proxy_html, mod_proxy_http, and mod_xml2enc

are NOT commented out (they usually are by default, so remove the "#" in front of the line):

```
[...]
LoadModule lbmethod_byrequests_module modules/mod_lbmethod_byrequests.so
[...]
LoadModule proxy_module modules/mod_proxy.so
[...]
LoadModule proxy_balancer_module modules/mod_proxy_balancer.so
[...]
LoadModule proxy_html_module modules/mod_proxy_html.so
LoadModule proxy_http_module modules/mod_proxy_http.so
[...]
LoadModule xml2enc_module modules/mod_xml2enc.so
[...]
```

Then make sure that the "Listen" directive is set to the port on which you want ARIS to be available (this is for HTTP access, HTTPS access is not in the scope of this guide), e.g.,

```
Listen 80
```

Also make sure that the DocumentRoot directive points to an existing directory from which you want to serve static content (we actually DON'T want our fake HA LB to serve static content aside from what is inside ARIS; but if that directive points to a non-existing or otherwise inaccessible directory, HTTPD will not start). For example, if your <apacheInstallDir> is "C:\Program Files\Apache\Apache24_fakeHWLB", the line should look like this (make sure that the htdocs subdirectory actually exists).

```
DocumentRoot "C:\Program Files\Apache\Apache24_fakeHWLB\htdocs"
```

Finally, add the following lines to the end of httpd.conf, which will make HTTPD load-balance all requests to our two ARIS LBs:

```
1 ProxyPass "/" "balancer://myaris/"
2 ProxyPassReverse "/" "balancer://myaris/"
3 ProxyPreserveHost On
4
5 <Proxy "balancer://myaris">
6   BalancerMember "http://arisworker1:80"
7       BalancerMember "http://arisworker3:80"
8 </Proxy>
9
10 <Location "/balancer-manager">
11   SetHandler balancer-manager
12 </Location>
```

Lines 5-8 define the "balancer", i.e., a representation of a group of worker nodes that can handle the requests of the application to which our "fake" HA LB is providing access:

- Line 5 names our balancer (here we name it "myaris" - this is an internal name and does not correlate to any hostname, etc.),

- Lines 6-7 enumerate the "BalancerMembers", i.e., the actual workers, which in case of an ARIS installation are the nodes containing ARIS LBs. Based on our example scenario, where we have two ARIS LBs, one on machine arisworker1 and another on arisworker3, we have two such entries. Each "BalancerMember" entry contains a URL pointing to the hostname and HTTP port of one ARIS LB.
  You must adjust these lines to the actual hostnames of your ARIS worker nodes (and you also must adjust the ports in case you configured the ARIS LBs to use a port other than port 80 for HTTP requests).

Line 1 tells Apache that all requests starting with "/" (i.e. ALL requests to our "fake" HA LB) should be handled by the balancer "myaris". (Do not forget the trailing slash after "myaris".) For more detailed documentation of the ProxyPass directive, see the Apache documentation page for mod_proxy[15].

Line 2 tells Apache to rewrite certain parts of a response so that they point to our "fake" HA LB instead of pointing to the worker. For more detailed documentation of the ProxyPassReverse directive, see the Apache documentation page for mod_proxy[16].

Line 3 makes our "fake" HA LB preserve the host used in client requests and pass it to the downstream server (i.e., our ARIS LBs).

---

[15]Apache Module mod_proxy documentation version 2.4 -
ProxyPass https://httpd.apache.org/docs/2.4/mod/mod_proxy.html#proxypass
[16] Apache Module mod_proxy documentation version 2.4 -
ProxyPassReverse https://httpd.apache.org/docs/2.4/mod/mod_proxy.html#proxypassreverse

Lines 10-12 add the balancer-manager application, which is accessible on the HA LB at <fakeHALBHostName>/balancer-manager.

After making these changes, start your Apache using httpd.exe in <apacheInstallDir>\bin.

After starting your system (which we will describe in the next chapter), you will be able to access your ARIS application through the "fake" HA LB, i.e. by using the URL http//aris.example.com (resp. the actual hostname of the machine on which you installed the "fake" HA LB) instead of using the hostnames of the worker machines containing ARIS LBs.

To check the state of your "fake" HA LB and the workers, access the balancer-manager application with a browser at http://<fakeHALBHostName>/balancer-manager (i.e., assuming your "fake" HA LB runs on aris.example.com, the URL is http://aris.example.com/balancer-manager). You should see an output like the following: figure 58
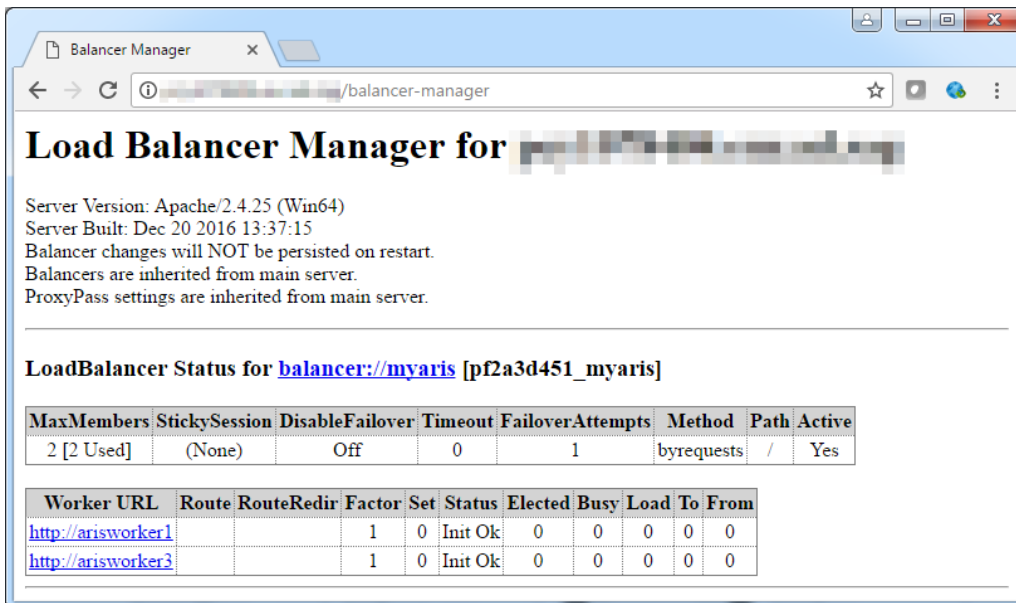
Observe that both workers are shown in Status "Init Ok".

You can then test the HA behavior by, e.g., shutting down one of the ARIS LBs. You will observe that clients will still be able to access ARIS without taking notice of the outage - only requests that were still active when the ARIS LB was shut down are interrupted, but the clients (both rich clients and browser-based thin clients) are built to tolerate this, in the worst case requiring you to reload the page. The balancer-manager application should then show the worker on which you stopped the ARIS LB in Status "Init Err":

FIGURE 59
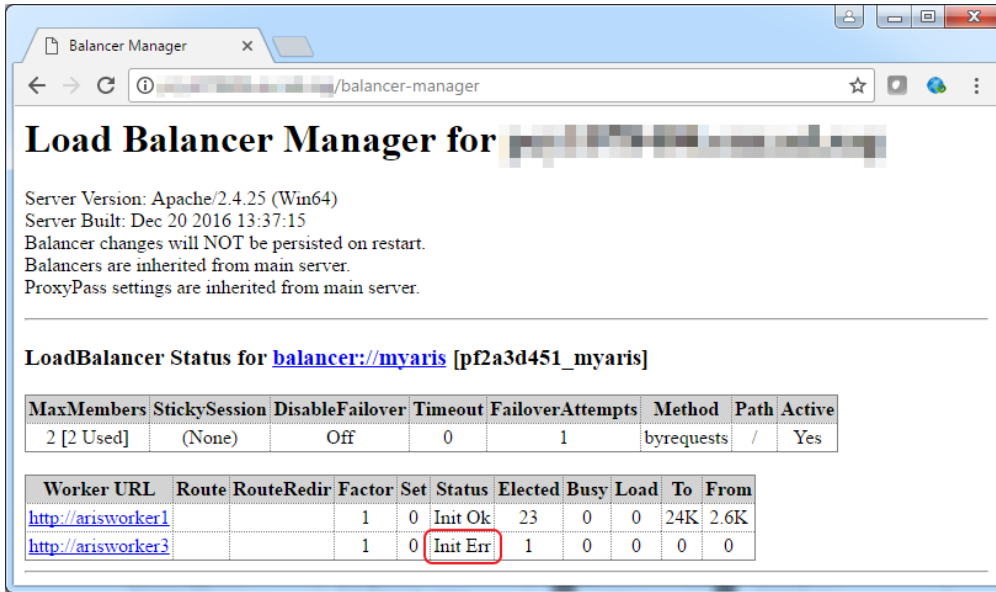
10   Administration of a distributed installation

In many aspects, the administration of a distributed installation does not differ too much from a single-node installation, in particular all administrative steps that do not directly pertain to the runnables or the nodes they reside on. For example, tenants can be created, deleted, backed up and restored using ACC (or the tenant management UI) as before - you must only make sure that all nodes that belong to the installation (and **only** nodes belonging to the installation, not those of other installations!) are registered in ACC or with the ACC server, as we described it in our chapter about the preparation steps for setting up a distributed system.

We also already explained the basics of how to use ACC to control multiple agents in another section of that same chapter.

10.1   Starting and stopping a distributed system

A single-node installation can be started and stopped using the startall and stopall commands on the node. You can use startall/stopall commands individually for all nodes of a distributed installation, e.g., by running "on n1 startall", "on n2 startall" etc. This however, is not only rather inconvenient, but it also has a significant drawback: while a simple startall (or stopall) command executed on a single node will preserve a certain order in which the runnables are started (or stopped, respectively), it will only preserve this order, among the runnables of **one** node. While the order in which runnables are started does not have to be observed as strictly as it was necessary in early versions of ARIS 9, there are still situations where violating the proper start order can lead to problems. While runnables should no longer fail to start up eventually when a service or other application they depend on is not available, they might still be stuck in STARTING state. For example, consider our distributed example scenario, shown once more in figure 60.

FIGURE 60: OUR DISTRIBUTED EXAMPLE SCENARIO
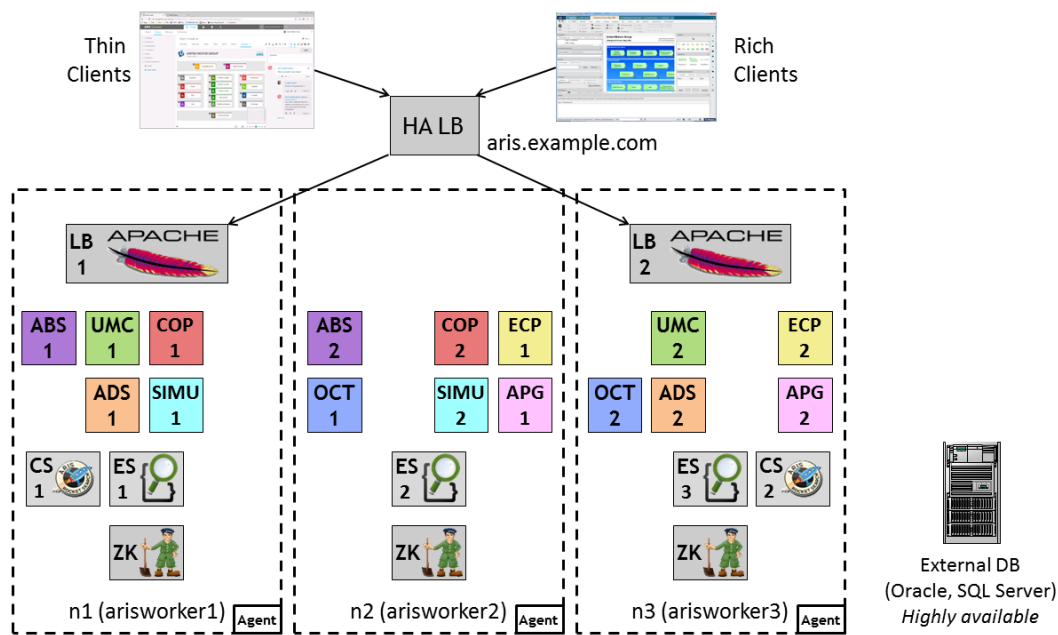
Assuming that all runnables of the installation are completely stopped, if we execute a startall command on node n1 first, this starts the Zookeeper instance on that node. Since the other two instances of our Zookeeper ensemble are still stopped, this instance cannot achieve the required quorum (2 out of 3 instances) and thus, while it will reach STARTED state, it will not server Zookeeper requests. Next, the "on n1 startall" command will proceed with starting Cloudsearch and Elasticsearch, which can reach STARTED state even without working Zookeeper access. Next runnable to be started is UMC, which, however, will wait for Zookeeper to become available (since it needs to do a Zookeeper lookup for its backend, i.e., Elasticsearch) and will be stuck in STARTING state, preventing the "on n1 startall" command from completing. Figure 55 below shows how this will apper in a second ACC console opened parallel to the running "on n1 startall" command.

FIGURE 61: THE PROBLEM OF USING "STARTALL" ON INDIVIDUAL NODES OF A DISTRIBUTED SYSTEM

```
ACC+ >on all nodes list

Node n1 - 9 installed runnables.
ID      State      Version  Type                                                              Extended state
zoo0    STARTED    10.0.0   com.aris.runnables.zookeeper-run-prod                             NOT SERVING
es0     STARTED    10.0.0   com.aris.runnables.elasticsearch-run-prod                         -
cs1     STARTED    10.0.0   com.aris.cip.y-cloudsearch-run-prod                               -
abs1    STOPPED    10.0.0   com.aris.modeling.components.y-server-run-prod                    -
umc1    STARTING   10.0.0   com.aris.umcadmin.y-umcadmin-run-prod                             -
cop1    STOPPED    10.0.0   com.aris.copernicus.copernicus-portal-server-run-prod             -
ads1    STOPPED    10.0.0   com.aris.adsadmin.y-adsadmin-run-prod                             -
lb1     STOPPED    10.0.0   com.aris.runnables.httpd.httpd-run-prod                           -
simu1   STOPPED    10.0.0   com.aris.modeling.components.simulationserver.y-simuserver-run-prod  -

Node n2 - 10 installed runnables.
ID      State      Version  Type
zoo0    STOPPED    10.0.0   com.aris.runnables.zookeeper-run-prod
es0     STOPPED    10.0.0   com.aris.runnables.elasticsearch-run-prod
pg      STOPPED    10.0.0   com.aris.runnables.PostgreSQL-run-prod
abs2    STOPPED    10.0.0   com.aris.modeling.components.y-server-run-prod
cop2    STOPPED    10.0.0   com.aris.copernicus.copernicus-portal-server-run-prod
ecp1    STOPPED    10.0.0   com.aris.runnables.ecp-run-prod
lb2     STOPPED    10.0.0   com.aris.runnables.httpd.httpd-run-prod
oct1    STOPPED    10.0.0   com.aris.octopus.y-octopus_server-run-prod
simu2   STOPPED    10.0.0   com.aris.modeling.components.simulationserver.y-simuserver-run-prod
apg1    STOPPED    10.0.0   com.aris.age.age-run-prod

Node n3 - 9 installed runnables.
ID      State      Version  Type
zoo0    STOPPED    10.0.0   com.aris.runnables.zookeeper-run-prod
es0     STOPPED    10.0.0   com.aris.runnables.elasticsearch-run-prod
cs2     STOPPED    10.0.0   com.aris.cip.y-cloudsearch-run-prod
umc2    STOPPED    10.0.0   com.aris.umcadmin.y-umcadmin-run-prod
ecp2    STOPPED    10.0.0   com.aris.runnables.ecp-run-prod
ads2    STOPPED    10.0.0   com.aris.adsadmin.y-adsadmin-run-prod
```

Note that the single started Zookeeper instance is shown with a "NOT SERVING" extended state (since it cannot get a quorum) and that the UMCAdmin instance on node n1 is stuck in STARTING state.

The proper start (stop) order must be maintained not merely among the runnables of the individual nodes, but across all runnables belonging to the installation. This is what the **on all nodes startall** and **on all nodes stopall** commands do: using information about the start order specified among all different runnable types (using the ACC config parameter "app.type.order"), these commands will start all instances of each app type across all nodes, before proceeding with instances of app types coming later in the app.type.order.

So in order to start (resp. stop) a distributed system (and assuming that the ACC was started with a node file, such that all nodes that belong to the distributed installation are known to ACC), you should use the **on all nodes** variant of **startall** (resp. **stopall**):

Starting a distributed application

```
on all nodes startall
```

Stopping a distributed application

```
on all nodes stopall
```

Like their single-node equivalents, these commands indicate the current progress of the operation, i.e., which runnables are currently being started (resp. stopped).

10.2   Collecting and deleting log files

Another operation that you might need eventually, in particular when you have to involve ARIS support to solve a problem with the system is to collect all log files of your distributed installation.

For quite some time now, you can use the **collectLogFiles** script that collects all log files on a single node. However, to run such a script, you must have access to each machine on the operating system level (for example log in remotely), then run the script, and retrieve the resulting ZIP file. This can be getting quite tedious when working with a multinode installation.

Since ARIS 9.8.6, ACC offers two new commands, "collect log files" and "delete log files", respectively, that allows you to conveniently collect (resp. delete) the log files of the agent and all runnables on individual nodes or on all nodes registered with ACC.

10.2.1   The "collect log files" command

The "collect log files" command enables you to collect the log files and certain configuration files of all runnables and the agent (and on Windows also the log files of the setup) on the current node, an explicitly specified node, or on all nodes, and put them into a ZIP file that is then placed on the machine running ACC.

The basic syntax of the command is

```
("on all nodes" | "on" <nodeName>)? collect log files ("of
<agentOrInstanceId> ("," <agentOrInstanceId>)*) ("to" <pathToResultFile>)?
```

You can run the command either on a specific node name (by prefixing it with the clause "on <nodeName>"), on all nodes (by prefixing it with "on all nodes)", or on the current node (by omitting either clause).

Further, you can optionally specify for which runnables and/or the agent you want to collect the log files by using the optional "of" clause. If the clause is omitted, the logs of the agent and all runnables on the selected node(s) are collected.

Finally, you can optionally specify the name of the ZIP file into which the log files should be placed by using the optional "to" clause. If the clause is omitted, the ZIP file will be placed in the current working directory of ACC.

Examples

```
collect log files
```

Collects the log files of all runnables and the agent (and on Windows also those of the setup) of the current node and puts them into a ZIP file located in the ACC's working directory.

```
on node n1 collect log files to c:\temp
```

Collects the log files of all runnables and the agent (and on Windows also those of the setup) of the node with the logical node name "n1" and puts them into a ZIP file located in the directory c:\temp.

```
on all nodes collect log files
```

Collects the log files of all runnables and the agent (and on Windows also those of the setup) of the all nodes currently registered with ACC and puts them into a ZIP file located in the ACC's working directory.

```
collect log files for agent, abs_m
```

Collects the log files of the agent (and on Windows also those of the setup) and the runnable named "abs_m" from the current node and puts them into a ZIP file located in the ACC's working directory.

10.2.2   The "delete log files" command

The "delete log files" command enables you to delete the log files of all runnables and the agent (and on Windows also the log files of the setup) on the current node, an explicitly specified node, or on all nodes.

Note that the log files of some runnables cannot be deleted when the runnable is not in STOPPED state, in particular the logs of the Postgres and the Loadbalancer runnable. Deleting these log files will be skipped and a warning will be issued.

The basic syntax of the command is:

```
("on all nodes" | "on" <nodeName>)? "force"? delete log files ("of
<agentOrInstanceId> ("," <agentOrInstanceId>)*)
```

You can run the command either on a specific node name (by prefixing it with the clause "on <nodeName>"), on all nodes (by prefixing it with "on all nodes"), or on the current node (by omitting either clause).
Further, you can optionally specify for which runnables and/or the agent you want to delete the log files by using the optional "of" clause. If the clause is omitted, the logs of the agent and all runnables on the selected node(s) are deleted.
Finally, you can optionally specify the "force" keyword to skip the security query that normally requires you to confirm the command. This is useful if you want to run the command from within scripts.

10.3   Updating a distributed installation

Refer to the ARIS Update Cookbook, which has extensive coverage of manually updating single-node and distributed, multi-node installations.
From ARIS 10.0.12.0, ARIS Risk & Compliance Manager and ARIS Server use the same external database management system if configured. When you update your ARIS Server, ARIS Risk & Compliance Manager still uses the database connection as configured for ARIS Risk & Compliance Manager. If you want ARIS Server and ARIS Risk & Compliance Manager to use the same connection, you must first update ARIS Server. Then configure the database connection manually, as described in the ARIS Risk & Compliance Manager Installation Guide.

11   Administration of a Heavy-Duty Server: Sandboxing for long running operations

The **hds** (Heavy-Duty Server) runnable is basically a specialized **abs** runnable for "sandboxing" long running tasks. However, hds does not contain ARIS Download Clients because hds does not provide any modeling tasks. It reduces, for example, the load of ARIS reports running on you **abs** instance. This has the advantage that the performance of the modeling/publishing/reporting server (**abs**) is no longer affected by the load of running reports or other tasks. If an **abs** runnable detects an **hds** runnable in the installation, it delegates report execution to **hds** by default.

You basically need to add two additional runnables to your installation, the **hds** runnable itself, and the **cdf** runnable (Cloud Decoupling Framework), which is required for communication between the components. The **cdf** runnable is a queue that can be used to decouple synchronous accesses to services.

For both runnables, the **hds** and the **abs**, the same requirements, such as database JDBC drivers and SAP Java Connector must be met. You can run hds on the same node as abs, but for optimum performance, you are recommended to use them on different nodes.

11.1   Installation

The **cdf** and **hds** runnables are not installed by default, but available as an optional installation add-on intended for heavy-load environments.

Start ARIS Cloud Controller (ACC) passing the **nodes.nf** file and the **generated.apptypes.cfg** file:

```
acc -n nodes.nf -c generated.apptypes.cfg
```

Make sure that you provided a remote repository from which the ARIS Agents can download the runnables to their local repositories (see: **Preparing and using a remote repository** on page **44**), for example:

```
on all nodes set
remote.repository.url="http://internal.company.com/arisrepo"
```

 You can also provide a file-based repository, for example:

```
on all nodes set
remote.repository.url=file://E:\\ARISDVD\\ARIS10\\Setups\\Windows\\Setup_Da
ta
on all nodes set local.repository.only=false
```

11.2 Heavy-Duty Server configuration

By default, an **abs** runnable only delegates report execution to an **hds** runnable.

abs runnable

You can make an **abs** runnable to also delegate other tasks, such as XML import/export, database operations (for example, merge, restore, or backup procedures), or tasks related to ARIS configuration (for example, consolidation or filter generation):

```
JAVA-Dcom.aris.modeling.server.HDUseCases="report,xml,config,db,portal"
```

> ⚠ **Important:** Each task type must be configured in the **hds** configuration to ensure that all tasks will be handled.

hds runnable

You can define which of the long-running tasks delegated by an **abs** runnable are to be handled by an **hds** runnable. You can also specify the number of tasks to be handled at the same time:

```
JAVA-Dcom.aris.modeling.server.HDMaxTasks="<numberOfConcurrentTasks>"
```

11.2.1   Fine-grained report execution (optional)

Optionally optimizes load balancing during script execution.

If complex scripts are executed in parallel, the system can slow down. The resource usage depends on the complexity of a script, for example, the number of server calls. In ARIS Architect, script administrators can select the expected resource usage of scripts. The values range from **Almost no resource usage**, **Low resource usage** (S), **Medium resource usage** (M is the default value for ARIS reports), **High resource usage** (L), and **Very high resource usage** (X). The selected resource usage value decides whether the script is executed on the **abs** runnable (Almost no resource usage) or on additionally installed Heavy-**duty** Servers (HDS).

The selected sizing tags (S, M, L, X) enable you to specify which reports are to be handled by an **hds** runnable. You can also forcet heavy-load reports (X) to be executed on one single **hds** runnable. The default configuration for runnables is " 'X,L,M,S' (all reports)":

```
JAVA-Dcom.aris.modeling.server.HDQueueTypes="X,L,M,S"
```

> ⚠️ **Important:** You do not generally need to change these settings. However, if you do, make sure that all report sizes are covered by all **hds** runnables**.**

11.2.2   Overload protection

You can configure **hds** runnables so that they accept tasks only if the heap memory is below a certain threshold:

```
JAVA-Dcom.aris.modeling.server.HDHeapThreshold=[20…99]
```

If the heap memory usage is too high, the **hds** instance waits for it to decrease sufficiently. This means that the **hds** runnable does not accept any new tasks as long as the heap memory uses more than 75% of the maximum possible memory usage (logged in trace.log):

```
JAVA-Dcom.aris.modeling.server.HDHeapThreshold=75
```

11.2.3   Maximum number of concurrent task executions

You can specify the number of tasks to be handled at the same time:

```
JAVA-Dcom.aris.modeling.server.HDMaxTasks="<numberOfConcurrentTasks>"
```

11.2.4   High-availability environments

Provide at least 2 **abs** runnables, 2 **cdf** runnables, and 2 **hds** runnables to ensure high availability (see page 16) . You are recommended to use the same options for related runnable types so that one runnable can replace another of the same type if a runnable fails.

This is mandatory also for performing rolling updates (see ACC command **drain**).

Example 1**: Basic configuration**

```
# Default  behavior : Only reports are delegated from an abs runnable to an
hds runnable.

on n1 configure abs_l abs_l

on n2 configure cdf_l cdf_l

on n2 configure hds_l hds_l
```

Example 2: Advanced configuration with specialized, extended work responsibility

```
# Configure an abs runnable in such way that it delegates other long-
running tasks ("report,xml,config,db,portal") to specialized hds runnables.

on n1 configure abs_l JAVA-
Dcom.aris.modeling.server.HDUseCases="db,config,xml,report,portal"

on n2 configure cdf_l cdf_l

on n2 configure hds_l hds_l " JAVA-
Dcom.aris.modeling.server.HDUseCases="db,config,xml,portal"

on n3 configure hds_l hds_l  JAVA-
Dcom.aris.modeling.server.HDUseCases="report"
```

Example 3: High-availability configuration (Recommended)

```
# High-Availability requires that all runnables are configured the same way
to allow each one to replace the other one

on n1 configure abs_l  JAVA-
Dcom.aris.modeling.server.HDUseCases="db,config,xml,report,portal"
on n2 configure abs_l  JAVA-
Dcom.aris.modeling.server.HDUseCases="db,config,xml,report,portal"

on n3 configure cdf_l cdf_l
on n3 configure hds_l hds_l   JAVA-
Dcom.aris.modeling.server.HDUseCases="db,config,xml,report,portal"

on n4 configure cdf_l  cdf_l
on n4 configure hds_l hds_l  JAVA-
Dcom.aris.modeling.server.HDUseCases="db,config,xml,report,portal"
```

1   Legal information

11.3   Documentation scope

The information provided describes the settings and features as they were at the time of publishing. Since documentation and software are subject to different production cycles, the description of settings and features may differ from actual settings and features. Information about discrepancies is provided in the Release Notes that accompany the product. Please read the Release Notes and take the information into account when installing, setting up, and using the product.

If you want to install technical and/or business system functions without Software AG's consulting services, you require extensive knowledge of the system to be installed, its intended purpose, the target systems, and their various dependencies. Due to the number of platforms and interdependent hardware and software configurations, we can only describe specific installations. It is not possible to document all settings and dependencies.

When you combine various technologies, please observe the manufacturers' instructions, particularly announcements concerning releases on their Internet pages. We cannot guarantee proper functioning and installation of approved third-party systems and do not support them. Always follow the instructions provided in the installation manuals of the relevant manufacturers. If you experience difficulties, please contact the relevant manufacturer.

If you need help installing third-party systems, contact your local Software AG sales organization. Please note that this type of manufacturer-specific or customer-specific customization is not covered by the standard Software AG software maintenance agreement and can be performed only on special request and agreement.

If a description refers to a specific ARIS product, the product is named. If this is not the case, names for ARIS products are used as follows:

Name	Includes
ARIS products	Refers to all products to which the license regulations of Software AG standard software apply.
ARIS Clients	Refers to all programs that access shared databases via ARIS Server, such as ARIS Architect or ARIS Designer.
ARIS Download clients	Refers to ARIS clients that can be accessed using a browser.

11.4   Data protection

Software AG products provide functionality with respect to processing of personal data according to the EU General Data Protection Regulation (GDPR).

Where applicable, appropriate steps are documented in the respective administration documentation.

11.5   Disclaimer

This document provides you with the knowledge for installing ARIS in distributed environments.

For acquiring that knowledge Software AG recommends that you attend the training course ARIS Server Installation available via http://softwareag.com/education, or ask for an ARIS installation service by Global Consulting Services.

ARIS products are intended and developed for use by persons. Automated processes, such as the generation of content and the import of objects/artifacts via interfaces, can lead to an outsized amount of data, and their execution may exceed processing capacities and physical limits. For example, processing capacities are exceeded if models and diagrams transcend the size of the modeling area or an extremely high number of processing operations is started simultaneously. Physical limits may be exceeded if the memory available is not sufficient for the execution of operations or the storage of data.

Proper operation of ARIS products requires the availability of a reliable and fast network connection. Networks with insufficient response time will reduce system performance and may cause timeouts.

If ARIS products are used in a virtual environment, sufficient resources must be available there in order to avoid the risk of overbooking.

The system was tested using scenarios that included 100,000 groups (folders), 100,000 users, and 1,000,000 modeling artifacts. It supports a modeling area of 25 square meters.

If projects or repositories are larger than the maximum size allowed, a powerful functionality is available to break them down into smaller, more manageable parts.

Some restrictions may apply when working with process administration, ARIS Administration, ARIS document storage, and ARIS Process Board, and when generating executable processes. Process Governance has been tested and approved for 1000 parallel process instances. However, the number may vary depending on process complexity, for example, if custom reports are integrated.

ARIS document storage was tested with 40.000 document items. We recommend monitoring the number and overall size of stored document items and archiving some document items if needed.