SUSE SAP automation solution

SUSE SAP automation solution

Peter Schinagl

Publication Date: 2020-12-03

SUSE LLC
1800 South Novell Place
Provo, UT 84606
USA
https://documentation.suse.com

*

Contents

*

*

I SUSE SAP automation solution

*

1 preface

This reference document contains best practices and planning considerations when using SUSE's Automation templates for SAP Landscapes. It is targeted at consultants and end-customers deploying SAP Landscapes in the public cloud and provides guidance on how the Terraforms, SALT, and other components work together to provide a consistent and validated architecture.

The document can also be used as a guide for a partner enablement workshop covering the proper use of these tools.

The following, layered [1] aspects will be covered:

- **Why** one should consider this strategy

- **Who** to engage with, inform and collaborate with

- **What** key factors are important and **When** to consider them

- **What** software and applications this is relevant to accomplish

- **How** various technology components can facilitate this

- **Where** the resulting solution may physically or virtually reside

- migration

---

[1] link: Archimate Enterprise Architecture (https://pubs.opengroup.org/architecture/archimate3-doc) ↗

2 Introduction

SUSE's Vision is to Simplify, Modernize, Accelerate the business of our customers.

Maintaining a competitive advantage often depends on how quickly new services can be delivered into a business. SAP applications are designed to help analyze data to anticipate new requirements, and rapidly deliver new products and services.

When SUSE first released SUSE Linux Enterprise Server for SAP Applications, it already included automated installation features for the SAP software stack. Over the last 10 years, the SUSE SAP LinuxLab and development engineers have introduced several additional features to automate routine system administration. Based on this experience, SUSE worked on reimagining the deployment wizard capability onto a modern framework.
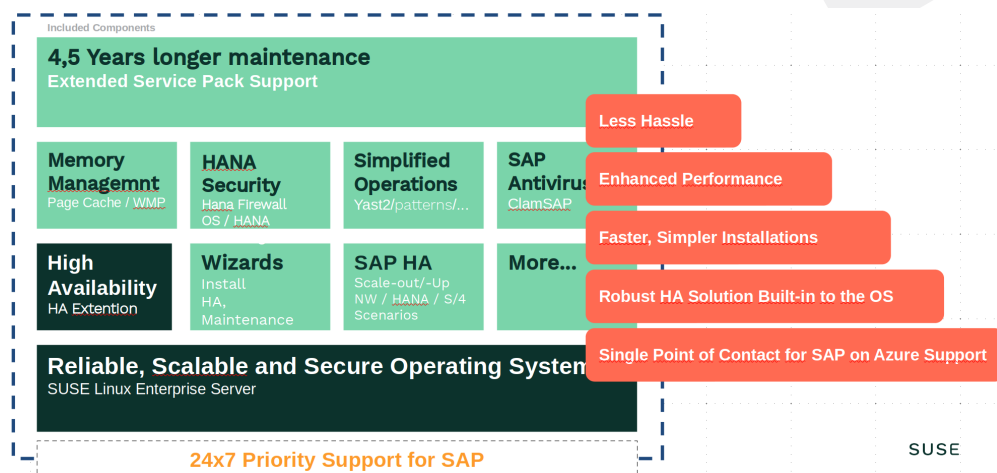
Simplify

    the deployment of an SAP Landscape in The Public Cloud for dev, test, and production.

Modernize

    customer environments by taking advantage of the benefits of the public cloud

Accelerate

    customer migrations to the cloud.

Building the infrastructure to run SAP Applications can be complex and demands significant effort, especially if a manual deployment method is used. When delivering muliple envionments, for mulitple engagements, reproducing the deployment can be tedious and error-prone.

One additional challenge is ensuring the infrastructure is highly available, as this will add more complexity and steps to the deployment.

When deploying and managing a large number of systems in an SAP Landscape, there is often a secondary need, getting an overview of what is happening in he environemt after the install is complete.

To help with this, SUSE has added the ability gain insight into your SAP Landscape with a comprehensive monitoring solution, providing dashboards, realtime and historic views and active alerts and reporting.

The major motivation was to improve, simplify and unify the installation of SAP Landscape on SUSE Linux Enterprise Server for SAP Applications and clearly standardize deployments and allow customers to use one level of tooling in various ways – from a Command Line interface, through some GUI driven process and SUSE Manager or other automation frameworks.

To achieve this, SUSE have adopted a more modern approach, infrastructure-as-code which helps customers reduce the effort and errors during deployments.

In recent years, SUSE Linux Enterprise Server and many other SUSE products shipped with a universal configuration management solution, this is used as the foundation for the new automation capability.

This configuration management system is called Salt from SaltStack and provides a highly scalable, powerful and fast infrastructure automation and management, built on a dynamic communication bus. Salt can be used for data-driven orchestration, remote execution for any infrastructure, configuration management for any app stack and much more.

Combining this configuration management system with an infrastructure deployment solution such as Terraform from Hashicorp makes it possible to do a hands free, error free setup of an SAP Landscape. Once the deployment is complete, administrators can login to start customizing the SAP System.

#As part of the deployment, SUSE added the ability provide insights into your SAP Landscape with comprehensive dashboards, realtime and historic views and active alerts and reporting based on flexible and powerful open-source projects Prometheus and Grafana.

The deployment automation can also be configured to setup a monitoring environment for the SAP HANA and SAP NetWeaver clusters.

*

3   Strategy

Most of the SAP services are deployed on premise with well known procedures, but you need to plan and size the needed hardware several years ahead and estimate the maximal workload e.g. for the next 5 years. You need to look at capacities, as also on other increasing demands and more or less guess this right values, because you do never know what will happen in the future.

In the current time, with quickly changing environments, many businesses have the need to accelerate innovation and increase agility across there business and a faster time-to-market with in addition lowering there costs.

With moving there products and services into the cloud, which is a actual modern trend in organizations, they can make there businesses meet the changing demand and to become flexible and agile in making business decisions.

As one example here, you do not need anymore plan the hardware size for the next 5 years, as a bigger and faster or smaller is only one reboot away. So you only need to do "rightsizing" as you can change the virtual hardware quickly - its only one reboot and you have changed the "hardware".

It is important that you have such changes in mind when migrating to the cloud, as not takeing care of such cloud benefits could be a huge cost factor - companies which do "rightsize" their workloads are able to cut costs by of 30% - 60%.

3.1   Context

There are quite many benefits by moving your SAP workloads to the cloud:

Quick deployment

If you need fast application implementation and deployment, cloud is the best choice. You can set up a cloud environment within a few hours, whereas, in-house IT infrastructure set-up takes days and sometimes month order it and to set up. With the cloud, IT teams can easily configure and manage the setup remotely.

It is a part where SUSE can help with our automation solution for SAP.

Reduce Cost

Many businesses spent a lot of their capital on maintaining their IT Infrastructure. With help of the cloud, they can transform their CapEx (Capital Expenditure) to stable OpEx (Operating Expenditure). The costs can be controlled since, the cloud model is a 'pay as

*

you use' scenario. Many cloud vendors provide a few more payment options to provide more choice to there customers. But this also means you need to adapt how you use the resources and the SAP software.

It is a part where SUSE can help together with the cloud provider to offer different payment options.

Scalability & Flexibility

With the cloud, you can scale up your system as and when required. Add or remove resources when the business demands it and what we have mentioned above with "rightsizing" too.

With SUSE ready to run images in the cloud providers marketplaces and infrastructure its easy to scale.

Maintenance

With the cloud, your IT department doesn't have to worry about managing and maintaining the hardware and parts of the infrastructure, the cloud service provider will handle it for your organization and you free up resources to focus on your business.

Resiliency

For businesses these days, uptime is of prime importance to run the daily business smoothly. Moving to the cloud maximizes uptime and reduces downtime. And the cloud could improve disaster recovery and business continuity without spend a huge amount of their capital on robust disaster recovery tool to save themselves from any security threat by using the offerings from the provider.

SUSE can help here with our SAP HA automation to help to provide less downtime of your application.

Remote access

With the cloud, your employees can access data from anywhere, and at any time via internet making business more flexible and increase the productivity of the employees.

SUSE products provides natively many options for remote access and control.

So overall, public clouds could offer significant benefits for all customers regardless of there size. The use of public cloud resources can lower there infrastructure costs and improve the scalability, agility, flexibility and availablity of applications.

4 Business

The Document is targeted at Consultants and End-customers who are deploying SAP Landscapes in the public cloud. Within Cloud environments there is no strict separation of responsible teams (like Network,DB,OS,Application) anymore as all can be done from a central place. But this should not make special knowledge not needed anymore. Such Teams still exist and still need to work together - best in a DevOps approach with Infrastructure-as-Code.

This mean you still need knowledge of the cloud infrastructure and there possibilities, as also details of the operating system and the tooling around like for HighAvailability. You also need to plan for the application usage and sizing.

SAP architectures needs to be fine-tuned based on the customer's requirement on system availability i.e 99.99% or 99.95% or 99.9%. So you need to identify the Single Point of Failures (SPOF) in the components and services. Such SPOF's then need to be in a HA cluster and all other infrastructure SPOF's need to have some degree of redundancy.

If you look at a typical SAP implementation you will find

1. SAP Central Services

2. a Database (e.g. HANA)

3. a Primary Application Server (PAS)

4. and some shared storage (NFS)

where the points 1,2 and 4 are potential SPOF's.

With our automation we try to eliminate all of these single-point-of-failures by providing HA cluster implementations to ensure automated failover, data protection and higher system availability.

SAP Central Services

> You need at least 2 VMs (Virtual Machines) to configure an ASCS/ERS high availability cluster. Depending on the SAP versions, you can configure the ASCS/ERS cluster in either ENSA 1 or ENSA 2 architecture which could be automated with the SUSE HA Extension (HAE).

Database layer

> You need at least 2 VMs (Virtual Machines) to configure SAP HANA HA/DR cluster in a scale-up deployment. The SUSE High Availability Extension is used to detect system failures and facilitate automatic failover.

*

Depending on the services used or available from the cloud provider it could be that you need a third cluster providing a High Available NFS service.

This is one of the main topics of the solution, that we are able to create all needed infrastructure and configurations to maximize the SAP System availablity.

4.1   What we cover with the automation

SAP HANA and Netweaver applications can be deployed in many different scenarios and combinations between them. So we have created building blocks which are modular and reusable to attend from single install to full cluster deployment.

As of today we take care of

- HANA single node

- HANA HA Scale-up Systemreplication

    - Performance Optimized Scenario

        - active/passive

        - active/readonly

    - Cost Optimized Scenario

- Netweaver

- Netweaver HA with Enqueue Replication Version (ENSA1)

- S/4 HANA

The solutions will get extended continuously depending on demands of customers and partners.

The overall landscape which get deployed look like this

*

SAP Architecture Overview

SAP S/4 HANA / Netweaver
with Enqueue-replication

SAP HANA Database
with System-replication

Monitoring Server

HA NFS Service
with DRBD

Split Brain Detector
(SBD)

Jumpbox

Copyright © SUSE 2020

SUSE
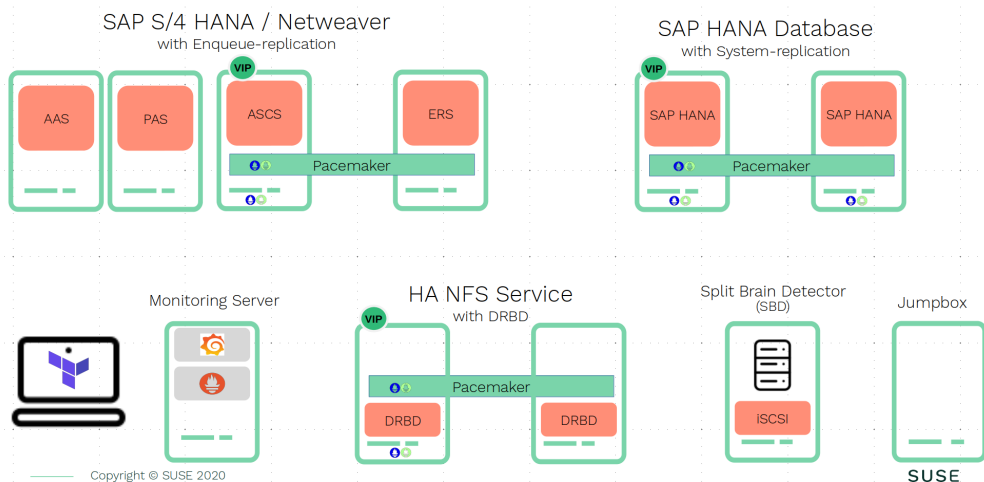
4.2   Prerequisites for SAP workloads in public clouds

There are a few general prerequisites to get a supported SAP Landscape in public cloud environments:

1. You need to have a SAP license for the SAP software to be deployed.

2. You need to know the resource requirements of your SAP workloads and have done a SAP system sizing

3. You need to use certified VM instance types and know the capacity required by SAP software

4. You need to provide the right network connectivity (bandwidth, latency, and package loss) within the cloud environment for your SAP workloads.

5. You need certified operating systems on which the SAP workloads will be deployed.

6. You should have knowledge of Linux and a of operation of SAP systems and the cloud infrastructure.

7. If there should be build high available systems, there need to be deep knowledge of the HA concepts and tooling and a good knowledge what is possible in the cloud infrastructure.

*

4.3   Roles and Collaboration

FixMe - Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Massa tincidunt nunc pulvinar sapien. Hendrerit dolor magna eget est lorem. Bibendum enim facilisis gravida neque convallis a cras semper auctor. Sit amet mattis vulputate enim nulla aliquet porttitor. Semper feugiat nibh sed pulvinar proin gravida. Tincidunt lobortis feugiat vivamus at augue eget arcu dictum varius. Tincidunt nunc pulvinar sapien et ligula ullamcorper malesuada proin. Pretium quam vulputate dignissim suspendisse in. Lectus proin nibh nisl condimentum id venenatis a. Neque aliquam vestibulum morbi blandit cursus risus at ultrices mi.

4.4   Processes and Functions

FixMe - Egestas egestas fringilla phasellus faucibus scelerisque. Aliquet porttitor lacus luctus accumsan. Ornare arcu odio ut sem nulla pharetra diam. Interdum velit euismod in pellentesque massa. Nulla at volutpat diam ut. Volutpat sed cras ornare arcu dui vivamus arcu felis bibendum. Ut faucibus pulvinar elementum integer. Urna cursus eget nunc scelerisque. Quisque sagittis purus sit amet volutpat consequat mauris nunc. Quis varius quam quisque id diam. Pretium aenean pharetra magna ac placerat vestibulum lectus mauris ultrices. Purus faucibus ornare suspendisse sed. Placerat orci nulla pellentesque dignissim enim sit amet venenatis. Gravida neque convallis a cras semper auctor neque vitae tempus. Vel pharetra vel turpis nunc eget. Facilisis volutpat est velit egestas dui id ornare arcu odio. Aliquam vestibulum morbi blandit cursus risus at ultrices mi.

4.5   Factors, Flavors and Deployment Types

FixMe - Id diam maecenas ultricies mi eget mauris. Cras fermentum odio eu feugiat pretium nibh ipsum consequat. Mi sit amet mauris commodo. Quam nulla porttitor massa id neque aliquam vestibulum morbi blandit. Pretium viverra suspendisse potenti nullam ac tortor vitae purus faucibus. Diam donec adipiscing tristique risus nec feugiat in fermentum. Dui id ornare arcu odio ut sem nulla. Eu sem integer vitae justo. Elementum eu facilisis sed odio. Ut morbi tincidunt augue interdum velit euismod in pellentesque massa. Gravida neque convallis a cras semper. Tellus at urna condimentum mattis. Faucibus purus in massa tempor nec feugiat nisl pretium fusce. Ut lectus arcu bibendum at varius vel pharetra vel turpis. Velit egestas dui id

*

ornare arcu odio ut sem. Faucibus interdum posuere lorem ipsum. In ante metus dictum at tempor commodo. Sed id semper risus in hendrerit gravida rutrum. Mauris pharetra et ultrices neque ornare aenean euismod. Sit amet tellus cras adipiscing enim eu turpis egestas pretium.

4.5.1 Factors

4.5.2 Flavors

4.5.3 Deployment Types

*

5 Application

5.1 SUSE Linux Enterprise Server for SAP Applications

Is the bundle product for SAP users based on SUSE Linux Enterprise Server and the High Availablity Extension with many helpers for running SAP Application.

5.2 SAP Application

In order to use the project some preliminary steps are required. One of them is to prepare the SAP installation software. The SAP software can be downloaded from https://launchpad.support.sap.com/#/softwarecenter ↗ and need to be done in manually before you can use the automation.

5.3 Cloud share for SAP Media

After downloading the needed SAP software, the files must be copied to a storage in the cloud to be accessable from the new installed virtual machines.

Azure offers shared storage (Azure Files) for applications using the Server Message Block (SMB) protocol which is a simple way to upload the SAP Media to it an use it from the installed machines for the SAP installation.

To use Azure Storage, you need to create first a storage account.

https://docs.microsoft.com/en-us/azure/storage/files/storage-files-introduction ↗

5.4 Terraform

Terraform is an open source tool created by HashiCorp for building, changing and versioning infrastructure.

As a infrastructure provisioning tool, it is responsible for creating the servers, but also load balancers, queues, monitoring, subnet configurations, firewall settings, routing rules, SSL certificates, and many other aspects of a infrastructure.

*

Terraform is more or less cloud-agnostic and allows a single configuration to be used to manage multiple providers, and to even handle cross-cloud dependencies. This simplifies management and orchestration, helping operators build large-scale multi-cloud infrastructures.

But it's not a tool to simple create a landscape with a press of a button in another cloud. The cloud providers don't offer the same types of infrastructure. For example the VMs, load balancers and other services offered by AWS, are very different as those in Azure and Google Cloud looking at the details of features and configuration or management or security or scalability, etc.

Terraform's approach is that you write code which is specific to a (cloud)provider and take advantage of the provider's unique functionality, but to use the same language and toolset for all providers. Therefor the natural name for such modules is "provider". So for example the *Azure Provider* can be used to configure infrastructure in Microsoft Azure using the Azure Resource Manager API's.

Configuration files describe to Terraform which components need to run for your application. Then you run the main terraform command to generate an execution plan which describes what Terraform will do to get to the planned desired state and then you execute this plan to create the described infrastructure.

We provide terraform configuration files for AWS, Azure, GCE and libvirt.

The open source version of the needed Terraform is shipped within the Public Cloud Module of SUSE Linux Enterprise Server for SAP Applications

In addition Azure provide an easy to access web based commandline (cloudshell) where Terraform is already pre-installed.

Azure provides different types of storage that are suitable for VMs that are running SAP workloads, so you should get familiar with the SAP requirements for Azure.

The suggestions from the terraform files for the storage configurations are meant as good directions to start with. But you still should analyzing the storage utilization patterns during runtime of the application. It could be the case that you realize that you are not utilizing all the storage bandwidth or IOPS provided. Therefore you might consider downsizing on storage or you will see the opposite way and your workload might need more storage throughput than suggested with these configurations. As a result, you might need to change the capacity, IOPS or throughput. Independent what you needs between storage capacity required, storage latency needed, storage throughput and IOPS required and least expensive configuration, Azure offers enough different storage types with different capabilities and different price points to find and adjust to the right compromise for you and your SAP workload.

5.5 SALT

SaltStack's configuration management system lets you define the applications, files, and other settings that should be in place on a specific system. The system is continuously evaluated against the defined configuration, and changes are made as needed.

- Salt works with so-called "States" which express the state a host should be in, using small, easy to read, easy to understand configuration files.

- The automation is written as "formulas" which are a collection of pre-written Salt States and Salt "Pillar" files.

- The Pillar files are the variables and data to build the system.

The good thing is that SLES-for-SAP Applications does ship all these tools as part of the product, so you can set up as you need.

Salt formulas can be applied in two ways:

Salt Master with Salt Client. All steps must be executed on the Salt Master machine.

Salt Client (Minion) only. All steps in must be executed in all of the Salt clients where the formulas are going to be executed, wh is the approach we use here within the framework as we do not need a central master system.

5.5.1 Netweaver

The Netweaver formula for bootstrapping and managing the SAP Netweaver platform takes care of:

- Extract the required SAP files for SAP Medias (.tar,.sar,.exe)

- and setting up

  - ASCS instance

  - ERS instance

  - PAS instance

  - AAS instance

  - Database instance (currently only HANA)

*

Besides that, the formula sets up all of the pre-requirements as:

- Hostnames

- Virtual addresses

- NFS mounts

- Shared disks

- SWAP partition space

The formula follows the best practices defined in the official SUSE documentation http://documentation.suse.com/sbp ↗

5.5.2 HANA

The HANA formula takes care of the following:

- Extract the required SAP files for SAP Medias (.tar,.sar,.exe)

- Installs SAP HANA.

- Apply "saptune" for HANA to configure and tune the OS for HANA usage

- Configures system replication.

- Preconfigure the High Availability cluster requirements.

- Configures the SAP HANA Prometheus exporter

5.5.3 HA

The HA bootstrap formula takes care of creating and managing a high availability cluster

- Creates and configures the High Availability cluster, like pacemaker, corosync, SBD and resource agents.

- Adjustments for the Azure Infrastructure

- Handle Netweaver, HANA and DRBD

*

Depending on the cloud requirements it may need an iSCSI server to be able to provide a shared disk for fencing where we use the iscsi-formula from SaltStack

The formula provides the capability to create and configure a multi node HA cluster. Here are some of the features:

- Initialize a cluster

- Join a node to an existing cluster

- Remove a node from an existing cluster

- Configure the pre-requirements (install required packages, configure ntp/chrony, create ssh-keys, etc)

- Auto detect if the cluster is running in a cloud provider (Azure, AWS, or GCP)

- Configure SBD (if needed)

- Configure Corosync

- Configure the resource agents

- Install and configure the monitoring ha_cluster_exporter

5.5.3.1 Other depended services

5.5.3.1.1 HA NFS Service

To build a HA NFS Service if there is none available, we can create one with help of 3 Linux services and the following

- DRBD formula

- HA formula

- NFS formula from SaltStack

5.5.3.1.2 iSCSI Service

The iSCSI-formula from SaltStack is able to deploy iSNS, iSCSI initiator, and iSCSI target packages, manage configuration files and then starts the associated iSCSI services.

5.5.3.1.3 NFS formula

A SaltStack formula to install and configure nfs server and client.

5.6 Monitoring

Starting from the idea of improving user experience, SUSE worked on how to monitor the several High Availability clusters that manage SAP HANA and SAP Netweaver in a modern way. For monitoring, we use the Prometheus toolkit and the Grafana project to visualize the data. To be able to monitor the clusters on either HANA or Netweaver we have written Prometheus exporters for it.

5.6.1 SAP HANA Database Exporter

The exporter provide metrics from more than one database or tenant. It provides

- Memory metrics

- CPU metrics

- Disk usage metrics

- I/O metrics

- Network metrics

- Top queries consuming time and memory

5.6.2 High Availability Cluster Exporter

Enables monitoring of Pacemaker, Corosync, SBD, DRBD and other components of High Availability clusters. This provides the ability to easily monitor cluster status and health.

- Pacemaker cluster summary, nodes, and resource status

- Corosync ring errors and quorum votes. Currently, only Corosync version 2 is supported.

- Health status of SBD devices.

- DRBD resources and connections status. Currently, only DRBD version 9 is supported.

*

5.6.3   SAP Host Exporter

Enables the monitoring of SAP Netweaver, SAP HANA, and other applications. The gathered metrics are the data that can be obtained by running the sapcontrol command.

- SAP start service process list

- SAP enqueue server metrics

- SAP application server dispatcher metrics

- SAP internal alerts

6 Technology

6.1 Terraform

With help of terraform and the modules for the cloudprovider we create the infrastructure for the SAP application and some supporting services.

The Terraform templates provide everything needed to create the right infrastructure components and provide in addition a lot of predefined possible choices to make it more simple to create the right virtual machines, disks, networks etc.

6.2 Salt

What is Salt?

Salt is a different approach to infrastructure management, founded on the idea that high-speed communication with large numbers of systems can open up new capabilities. This approach makes Salt a powerful multitasking system that can solve many specific problems in an infrastructure.

The backbone of Salt is the remote execution engine, which creates a high-speed, secure and bi-directional communication net for groups of systems. On top of this communication system, Salt provides an extremely fast, flexible, and easy-to-use configuration management system called Salt States.

Our SAP landscape is made up of groups of machines, each machine in the group performing a role. Those groups of machines work in concert with each other to create an application stack.

To effectively manage those groups of machines, an administrator needs to be able to create roles for those groups. As example, a group of machines that serve front-end web traffic might have roles which indicate that those machines should all have the webserver package installed and that the web service should always be running.

In Salt, the file which contains a mapping between groups of machines on a network and the configuration roles that should be applied to them is called a *top file*.

Top files are named *top.sls* by default and they are so-named because they always exist in the "top" of a directory hierarchy that contains state files. That directory hierarchy is called a state tree and this is what we use to reference the building blocks for the SAP Landscape.

*

6.2.1   SAP Sizing

To make the SAP sizing more simple we have introduced pre-defined sizes with well known abbreviation from T-Shirt sizes Small (S), Medium (M), Large (L).

The Small (S) size is thought for non productive scenarios, where as the Medium (M) and Large (L) sizes for production setups, where certified instance types being used.

But you can also tweak such size or do complete your own settings.

As sizing is important, you need to choose the right instance sizes from the cloud provider which are certified by SAP and choose the right number of disks to provide the right I/O, as similar the network throughput for the solution.

6.2.2   Building Blocks

Our main building blocks for an SAP Landscape are the *Application Layer*, based on Netweaver with SAP Central Services (xSCS) with an Primary Application Server (PAS) and Additional application server (AAS) and the *Database Layer*, which is in our case SAP HANA.

There are two possible models how you can deploy SAP Business Suite, a centralized deployment where all runs on one server or distributed deployment where every service has its own node.

The centralized deployment is mostly used for non-production environments such as sandbox and development environments.

The distributed deployment, is the recommend way for production environments where each of the SAP application layer components is independently installed on different instances. Which is the scenario we have choosen for the automation.

We use the needed building blocks to build this landscape within the top.sls file and depending on the role of the node, the needed actions are triggered.

6.2.3   High Availability

The two main building blocks could be made high available in order to have less downtime and eliminate single point of failures. As there are many scenarios available from SAP, we support the most used scenarios, with the solution.

For the Central Services this is Enqueue Replication and for the database its the HANA System-replication.

This mean we need to create one additional machine for the building blocks to build a two-node cluster within the HA Scenarios.

To provide something like a "virtual IP address" which is able to move between the two cluster nodes, we use the *Standard Load Balancer* service from Azure to provide traffic to only the active node.

Frontend **10.0.0.13**
hn1-db
Probe Port **62503**

Availability set

HANA node 1
hn1-db-0
10.0.0.5

HANA System Replication

*
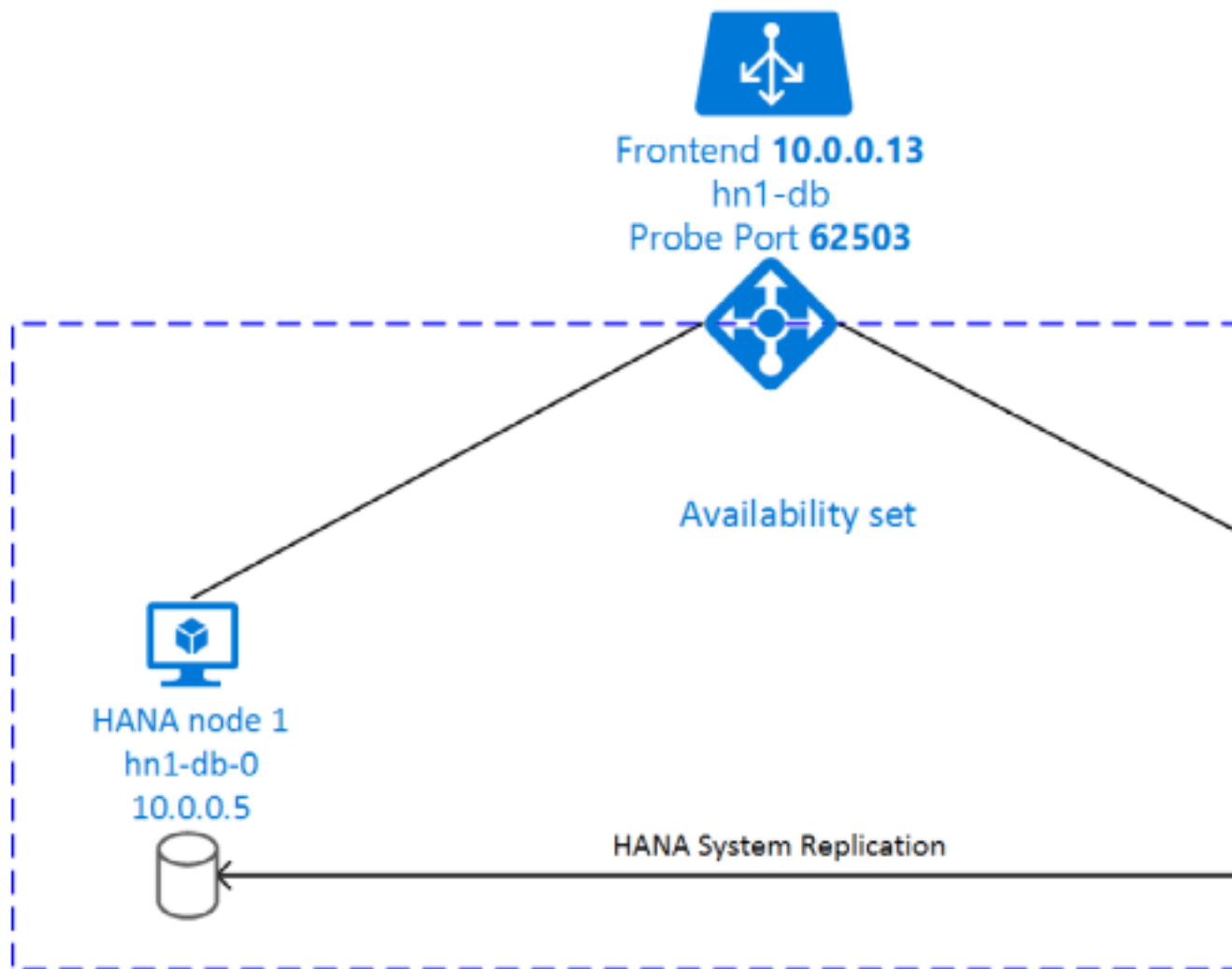
6.2.4   Additional Services

Depending on offered services from the cloud provider, we may need to build some services by our own if they are not there or use whats available, and similar to the above steps its reflected within the top.sls file.

6.2.4.1   NFS service

As we started with Azure, there was no NFS service available, so we need to build some with the tools we ship in SUSE Linux Enterprise Server for SAP Applications. As the NFS service should be high available, we need a second virtual machine to build a two node cluster.

*
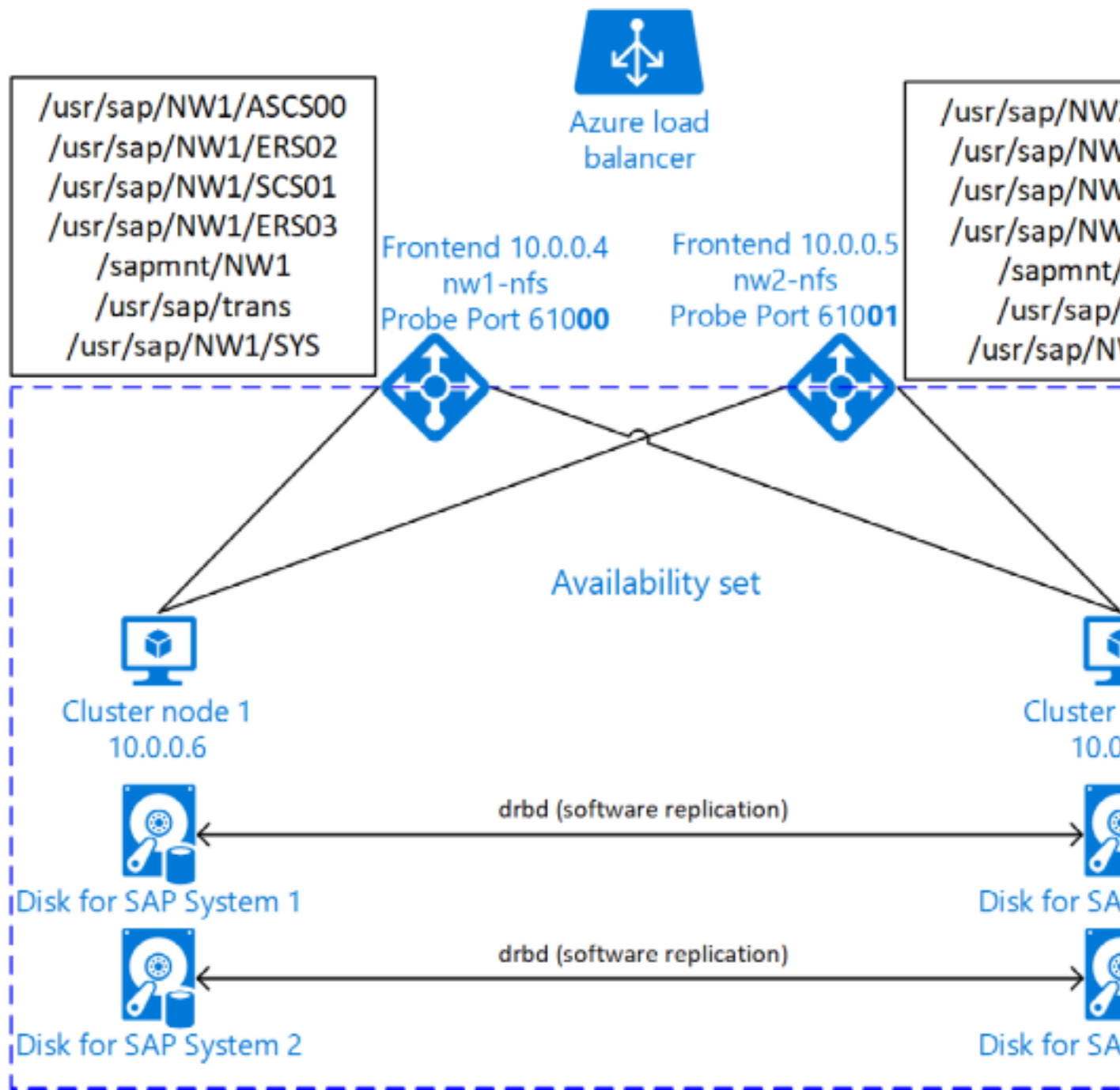
Over the time, Azure provide more and more services. So as of time of writing, there is a native NFS service with help of Netapp available (Azure Netapp files - ANF) and the Azure file service is extending in this direction too.

If the cloud native NFS service is used, not additional virtual machines will be created and the native service need to be set up in forehand.

6.2.4.2 Fencing service

For high available clusters, we need a mechanism to switch off one machine in the case of a so called "split-brain", mean when they can not communicate with each other anymore.

There are several methods which can be used and it depends again of the possibilities of the infrastructure.

As we started with Azure, Microsoft and SUSE created a fencing agent for the cluster. Such a fencing agent should be remove a machine as fast as possible (immediate) from the cluster, to make sure that there is only one active node, in order to avoid data corruption.

At this point of time, the Azure infrastructure provided only a way to graceful shutdown a machine, which took 10-15 minutes - which is by far too long for the split-brain case.

We need to create our own mechanism to fence machines. One technology we provide within our HA Extension is, using storage as additional communication between the nodes for such a split-brain case. This needs a raw shared disk, in order that both nodes can write messages to a central place. It is called SBD - Stonith Block Device or Split Brain Detector.

Unfortunately the Azure infrastructure did, at this point, not provide such a raw disk service, so we need to build it with the Linux tools we have in the distribution. With help of a iSCSI server, we can provide a raw shared disk within the cloud and therefore we are able to use the SBD fencing method which with help of the linux watchdog mechanism provides a fast and reliable fencing tooling.

This mean one additional server to provide a iSCSI service.

*
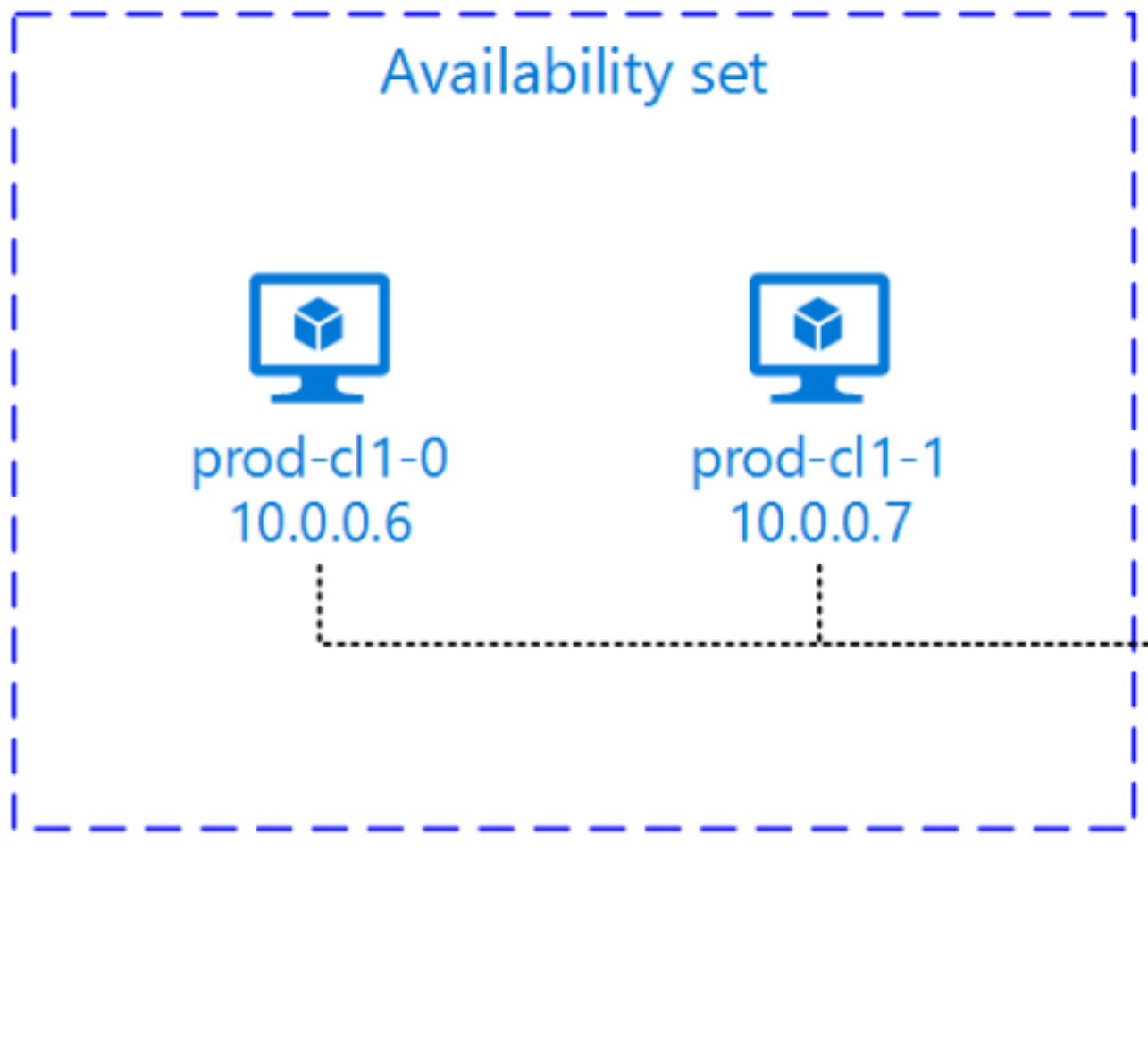
In the meantime there is a way in the Azure API to "kill" a virtual machine, so that the fencing agent can make use of it and no additional machine is needed if the fence agent is used. The drawback here is, in order to talk to the Azure API, there is a need for public network connection.
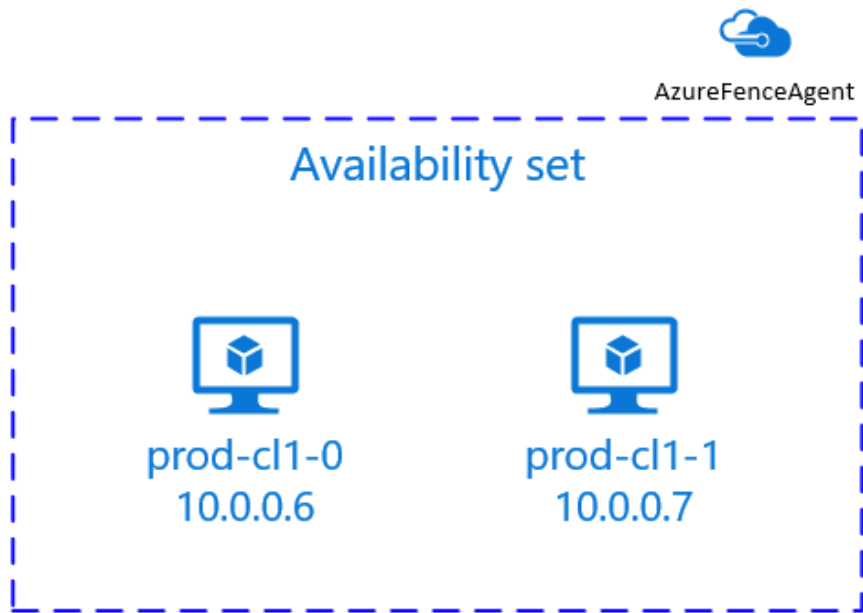
So you can choose between two ways . SDB fencing with help of an iSCSI service . agent based fencing through API access

6.2.4.3   Monitoring service

If you decide to use also the monitoring we provide with help of Prometheus and Grafana, we need a additional virtual machine for this service.

7 Physical

The SAP automation consist, as we have see before, out of several building blocks. First we want to look at the infrastructure deployment with terraform and then to the salt part.

As the project is under activ development to make it better and simpler to use, this document focus on the project version 6.0.0 of the terraform part and v6 of the rpm packages for salt formulars. New version could have more features or slightly changed files as show here, but the general guidelines provided here are still valid. ---

7.1 First make sure that all **pre-requirements** are done:

1. Have an Azure account

2. Have installed the Azure commandline tool *az*

3. Have installed *terraform* (v12) (it comes with SLES within the public cloud module)

4. Have the SAP HANA install media downloaded from SAP

5. Have created an Azure File Share

6. Copy or write down the the name of the storage account and the storage key, which is similar to a password.

7. Copy the SAP HANA install media to the Azure fileshare

8. extract the install media

As suggestion, we recommend the following directory structure:

```
share
  ├─ HANA
  │    └─ ...
  └─ Netweaver
       ├─ sapkernel
       │    ├─ ...
       ├─ export
       │    ├─ ...
       ├─ client
       │    ├─ ...
       └─ swpm
```

*

```
        ├── ...
```

7.2 Get the project from the github site

The project is hosted on a public github site where you can download it to your local machine.

After moving into this directory you will see the following directory structure:

```
├── aws
├── azure
├── doc
├── gcp
├── generic_modules
├── libvirt
├── LICENSE
├── pillar
├── pillar_examples
├── README.md
└── salt
```

The directories with the names of the *cloud provider* (aws,azure,gcp,libvirt) are the terraform templates for the relevant provider.

The *doc* directory has some short but important documents for certain parts of the solution

The directory *generic_modules* provides modules which get used by all cloud vendor templates, like common variables, local executed functions within the building block, dependent actions on destroy and the functions to start the SALT execution on the module building blocks.

The other directories *pillar, pillar_examples* and *salt* contain the part of the salt configuration management.

7.3 Terraform Building Blocks

Terraform relies on so called "providers" to interact with remote systems.

Providers are plugins and released independent from Terraform itself, this mean that each provider has its own series of version numbers.

Each Terraform module must declare which providers it requires, so that Terraform can install and use them.

If we switch to into a *cloud provider* directory, we see one directory *modules* and several *.tf* files which all together build the terraform template.

When creating Terraform configurations, it's best practice to separate out parts of the configuration into individual .tf files. This provides better organization and readability.

```
├── infrastructure.tf
├── main.tf
├── modules
├── outputs.tf
├── README.md
├── terraform.tfvars
├── terraform.tfvars.example
└── variables.tf
```

The *infrastructure.tf* file

provides the cloud specific setup with the cloud relevant provider module of terraform and defines all the needed cloud specific entities.

The *main.tf* file

provides all the values for the various variables needed for the modules. It is the main entry point for terraform.

The *modules* directory

provides more subdirectories which are the nested child modules which represent our technical building blocks

The *output.tf* file

provides the values handed out from the modules, e.g to be used or displayed.

*terraform.tfvars_*

is a variable definitions file which get automatically loaded instead of providing values manually and is the main file and maybe the only terrafrom file which should get changed.

terraform.tfvars.example

is a example file with many pre-filled key-value pairs to set up the solution. **You can use this as starting point for your own file.**

variables.tf

provides all used input variables, including a short description, the type of the variable and a default value which can be overwritten with the terraform.tfvars file. Please have a deep look at all variable and the comments for it, to get aware whats is possible.

e.g. the variable provisioner is like a switch if the salt part should run or only the terraform part.

A *module*

is a container for multiple resources that are used together. Modules can be used to create lightweight abstractions, so that you can describe your infrastructure in terms of its architecture, rather than directly in terms of physical objects.

We use the modules as our technical building blocks e.g. a HANA node and the module directory consist again *main.tf*, *variables.tf*, *outputs.tf*.

These are the recommended filenames for a minimal module, even if they're empty. *main.tf* is the primary entrypoint how to build the infrastructure building block.

There is one additional file (salt_provisioner.tf), which take care of handing over the needed values to the salt building blocks, with help of a special terraform resource called *null_provider*, and *triggers* the remote execution of salt after terraform has created the node with *connections* to the new created machines. It now starts the *provision* of the salt part. It also create user-defined variables for salt as grains file (/tmp/grains) for the respective module building block

7.4   Simple Install

We provide a example terraform and example pillar files to provide a very easy start.

1. Open a browser and goto https://github.com/SUSE/ha-sap-terraform-deployments ↗

2. Click on *tags*

3. Click on *6.0.0*
   You see what's new and what has changed - so if you use older versions make sure you read it carefully.
   The *Usage* section provides you with a link to a OpenBuildServer (OBS) repository where the RPM packages of the above discussed building blocks are stored which fit to the project version.
   You need to use this value within the terraform variables file. So copy the line as described

4. Now go to *Assets* and download the *Source code* as .zip or .tar.gz

5. Extract it into a folder on your computer

6. Goto this folder and into the sub folder *azure*

7. Copy the file *terraform.tfvars.example* to *terraform.tfvars.example* You will see many key-value variable pairs, some enabled some disabled with a = in front. To have a simple start, only touch what we describe below

*

8. So you need to change the region where you want to deploy the solution, so change *az_region = "westeurope"* to the azure region you want to use

9. To make it more easy to start, please change all 4 images types to pay-as-you-go (PAYG) to do so replace all *offer* settings with "sles-sap-15-sp2" and *sku* with 15
   Do this for hana, iscsi, monitoring, drbd e.g.

   ```
   hana_public_offer     = "SLES-SAP-BYOS"
   hana_public_sku       = "12-sp4"
   ```

   with

   ```
   hana_public_offer = "sles-sap-15-sp2"
   hana_public_sku   = "gen2"
   ```

   This will make use of the on-demand images which have automatically all needed SUSE repositories attached.

10. Next is to set the name of the *admin_user* to a name which you want to use

11. The next step is to provide ssh keys to access the machines which will be deployed.
    We recommend to create new keys for this as you need to provide both keys as they need to get copied to the machines. So change the two locations variables and point them to your files.

12. As we need SAP Install Media for the automatic deployment of HANA, you need to create a azure storage account where you need to copy the HANA media. Best would be if you already have extracted the SAP media to save time during the deployment.
    Then we need to provide the name,key and the path to this storage account to the system. So change

    ```
    storage_account_name
    storage_account_key
    hana_inst_master
    ```

    The inst_master variable should point to the directory where you have the extracted hana install files. There are more possibilities, but for the simples usage have everything already extracted on you share
    So disable the other hana variables with adding a '#' in front of them

    ```
    #hana_archive_file = "IMDB_SERVER.SAR"
    #hana_sapcar_exe = "SAPCAR"
    ```

*

```
#hana_extract_dir = "/sapmedia/HDBSERVER"
```

13. We need additional ssh keys for the cluster communications, so please save your changes and run the following commands from the azure directory

```
mkdir -p ../salt/hana_node/files/sshkeys
ssh-keygen -t rsa -N '' -f ../salt/hana_node/files/sshkeys/cluster.id_rsa
```

14. Please open the tfvars file again as we need a few final changes.
To create a HANA Systemreplication HA automation uncomment

```
#hana_ha_enabled = true
```

by removing the #
As now the system creates a cluster, we need to enable a few other services. Uncomment

```
#hana_cluster_sbd_enabled = true
```

by removing the #

15. Now we need to point the place where the right packages for the v6 could be found. Copy the variable from step 1 e.g.

```
ha_sap_deployment_repo = "https://download.opensuse.org/repositories/network:ha-clustering:sap-deployments:v6"
```

16. If you want the additional monitoring be deployed, simply uncomment

```
#monitoring_enabled = true
```

17. As last step we enable a simplification parameter which try to find out a few settings automatically. So scroll down to the end and uncomment

```
#pre_deployment = true
```

Now we have all settings for terraform done and are nearly at the step to run the deployment, so save your changes.

*

1. go one directory up and change into the *pillar_example* directory and here into the *automatic* directory where you can see 3 further directories. They will provide the configuration variable for the relevant services. This automatic folder will work for all cloud providers we support today, but is more complex as it normally need to be.

2. As we use only HANA, please switch to the *hana* directory and open the file *hana_sls*. The file looks quite complex - but we only need to change a few settings. Normally you would do provide a more simple file with your dedicated settings, but as we want to do it automatic, we use this file.

3. We need to change the PRIMARY_SITE_NAME with some name you want to set and also the name for the SECONDARY_SITE_NAME. You can change other settings like the passwords, but for a simple test you can leave it. So save you changes and go back to the main directory.

4. now we are ready to run terraform

```
az login
terraform init
terraform workspace new yourprojectname
terraform plan
terraform apply
```

If all goes well you will have after ~40 Minutes a installed and running HANA Systemreplication Cluster in Azure

As there is a jumphost with a public ip adress is created you simply can login to all machines from your machine with

```
ssh -J <adminuser>@jumphost <adminuser@targethost>
```

7.4.1 Terraform file details

All files in the Terraform directory using the .tf file format will be automatically loaded during operations.

The *infrastructure.tf* provides the *data sources* for the network setup, which are computed in other terraform parts and some *locals* variables used for mainly for the autogeneration of the network. In addition it provides the *resources* for the network setup with virtual network, the needed subnet and routing, the needed resourcegroup to be used, a storage account, the all the network security groups (nsg) being used and defines the jumphost.

The *main.tf* file is our main file and calls the child modules which consist of our building blocks and the required input and output variables defined by the child module. It in addition provides the calculation for the autogenerated ip addresses.

There is the (default) possibility to autogenerate network addresses for all nodes. For that you need to remove or comment all the variables related to the ip addresses (more information in variables.tf). With this approach all the addresses will be retrieved based in the provided virtual network addresses range (vnet_address_range).

TABLE 7.1: AUTOGENERATED ADDRESSES EXAMPLE BASED ON 10.74.0.0/16 VNET ADDRESS RANGE AND 10.74.0.0/24 SUBNET ADDRESS RANGE

Name	Terraform variable	IP Address	Comment
iSCSI server	iscsi_srv_ip	10.74.0.4	needed for SBD device in HA configuration
Monitoring	monitoring_srv_ip	10.74.0.5	if monitoring is enabled
HANA IP's	hana_ips	10.74.0.10, 10.74.0.11	second only used in HA
Hana cluster virtual IP	hana_cluster_vip	10.74.0.12	Only used if HA is enabled in HANA
Hana cluster virtual IP secondary	hana_cluster_vip_secondary	10.74.0.13	Only used if the Active/Active HA setup is enabled
DRBD IP's	drbd_ips	10.74.0.20, 10.74.0.21	needed if HA NFS service for NW is used
DRBD cluster vIP	drbd_cluster_vip	10.74.0.22	needed if HA NFS service for NW is used

*

Name	Terraform variable	IP Address	Comment
Netweaver IP's	netweaver_ips	10.74.0.30, 10.74.0.31, 10.74.0.32, 10.74.0.33	Addresses for the ASCS, ERS, PAS and AAS. The sequence will continue if there are more AAS machines
Netweaver virtual IP's	netweaver_virtual_ips	10.74.0.34, 10.74.0.35, 10.74.0.36, 192.168.135.37	The 1st virtual address will be the next in the sequence of the regular Netweaver addresses

If you want to use already existing network resources (virtual network and subnets) it can be done by configuring the *terraform.tfvars* file and adjusting the responsible variables.

The example of how to use them is available at *terraform.tfvars.example*.

Important

If you are specifying the IP addresses manually, make sure these are valid IP addresses. They should not be currently in use by existing instances. In case of shared account usage in cloud providers, it is recommended to set unique addresses with each deployment to avoid using same addresses.

The *output.tf* file is a way to expose some of the internal attributes, and act like the return values of a terraform module to the user. It will return the IP address and node names created from the automation.

The values defined in the *variables.tf* file are used to avoid hard-coding parameters and provides all needed terraform input variables and there default values within the solution instead of having them in the main.tf file.

As we have many variable values to input, so we define them in a variable definition file named *terraform.tfvars* and terraform will automatically load the variable values from the variable definition file if it is named terraform.tfvars

The *modules* directory provide all the needed resources to create the respective building block

```
modules/
├── bastion
│   ├── main.tf
│   ├── outputs.tf
│   ├── salt_provisioner.tf
│   └── variables.tf
├── drbd_node
│   ├── main.tf
│   ├── outputs.tf
│   ├── salt_provisioner.tf
│   └── variables.tf
├── hana_node
│   ├── main.tf
│   ├── outputs.tf
│   ├── salt_provisioner.tf
│   └── variables.tf
├── iscsi_server
│   ├── main.tf
│   ├── outputs.tf
│   ├── salt_provisioner.tf
│   └── variables.tf
├── monitoring
│   ├── main.tf
│   ├── outputs.tf
│   ├── salt_provisioner.tf
│   └── variables.tf
├── netweaver_node
│   ├── main.tf
│   ├── outputs.tf
│   ├── salt_provisioner.tf
│   └── variables.tf
└── os_image_reference
    ├── outputs.tf
    └── variables.tf
```

The respective file *salt_provisioner.tf* set the ***role*** of the **node** and handover the needed variables which where set in terraform, **as custom Salt *grains* for the node** with help of a terraform file provisioner and starts the salt provisioning process via .

7.4.2   SAP Sizing

One of the very important points to consider of a SAP deployment is Sizing and applies across three key areas: compute power, storage space and i/o capacity and network bandwith.

If this is a greenfield deployment, please use the SAP Quick Sizer tool to calculate the SAP Application Performance Standard (SAPS) compute requirement and choose the right instance types which have the closest match to the performance needed.

If you have an SAP system running that you want to extend with new functionality and/or add new users or migrate to SAP HANA perform brownfield sizing.

Overall it is an iterative and constant process to translate your business requirements to the right (virtual) hardware resources.

This is a mandatory step and should not be underestimated.

If you have some performance number, we want to make it easier to deploy the right instance sizes with the right disks types and performance, and the right network settings, we introduced a simplified SAP sizing which well known T-Shirt sizes, S,M,L and a very small Demo size.

Behind the sizes, are useful combinations to provide certain SAP performance scenarios. Below is a simple reference of the possible performance values

- Demo

- Small < 30.000 SAPS

- Medium < 70.000 SAPS

- Large < 180.000 SAPS

You can simply customize the used settings within the terraform.tfvars, or more permanent in the variables file.

The Demo and Small size are thought for non-production scenarios and do not use SAP certified instancetypes, whereas the Medium and Large are meant for production usage and therefor use SAP certified instance types. The setups also used the right disks and I/O behavior for production.

The SAPS values are meant for the landscape and not only for the database.

7.4.2.1 HANA

Given that low storage latency is critical for database systems, even for in-memory systems as SAP HANA. The critical path in storage is usually around the transaction log writes of the DB systems, but other operations like savepoints or loading data in-memory after crash recovery can be critical.

Therefore, it is mandatory to leverage Azure premium storage or Ultra disk for /hana/data and /hana/log volumes. Depending on the performance requirements, we may need to build a RAID-0 stripe-set to aggregate IOPS and throughput to meet the application scenario need.

The overall VM I/O throughput and IOPS limits need to kept in mind when deciding for a instance type.

Actual recommendations could be looked at the following link https://docs.microsoft.com/en-us/azure/virtual-machines/workloads/sap/hana-vm-operations-storage↗

The maps below, describes how the disks for SAP HANA will be used and created during the provisioning.

disks_type

   as HANA has high I/O requirements the disk type Premium SSD need to be used

disks_size

   is the size of the additional disk in GB, as every size has certain IOPS caps

caching

   The caching recommendations for Azure premium disks are assuming the I/O characteristics for SAP HANA /hana/data - no caching or read caching /hana/log - no caching - exception for M- and Mv2-Series VMs where Azure Write Accelerator should be enabled /hana/shared - read caching

writeaccelerator

   Azure Write Accelerator is a functionality that is available for Azure M-Series VMs exclusively. As the name states, the purpose of the functionality is to improve I/O latency of writes against the Azure premium storage. For SAP HANA, Write Accelerator is supposed to be used against the /hana/log volume only. Therefore, the /hana/data and /hana/log are separate volumes with Azure Write Accelerator supporting the /hana/log volume only.

Number of Disks

   The number of disks which get used, depend on the performance requirements. We join disks to a stripe set to provide more performance. Minimal we need 4 to 5 disks.

LogicalVolumes

   We are using LVM to build stripe sets across several Azure premium disks. These stripe sizes differ between /hana/data and /hana/log and the recommendations is 256 KB for /hana/data 64 KB for /hana/log

Name of the VolumeGroup

*

The name of the volume group used

Mount path

The mount point where the volume gets mounted

The number of elements **must match** in all of them

 # character

is used to split the volume groups

, (comma)

is used to define the logical volumes for each volume group

The number of groups splitted by "#" **must match** in all of the entries

names

The names of the volume groups (example datalog#shared#usrsap#backup#sapmnt)

luns

The luns or disks used for each volume group. The number of luns must match with the configured in the previous disks variables (example 0,1,2#3#4#5#6)

sizes

The size dedicated for each logical volume and folder. Example 70,100#100#100#100#100

paths

Folder where each volume group will be mounted. Example /hana/data,/hana/log#/hana/shared#/usr/sap#/hana/backup#/sapmnt/

The values could be set with the variables "hana_vm_size", "hana_enable_accelerated_networking" and "hana_data_disks_configuration" in the *variables.tf* file if you want to change the default (demo) or better in the *terraform.tfvars* to set actual values.

7.4.2.2 Netweaver

NetWeaver is SAP's integrated technology platform and is not a product in itself, but it provides the needed services for the SAP business applications and always need a database to talk to.

So it's the overall task of sizing need to take care of Netweaver plus the Database and this is what we combined with the T-Shirt sizes of the solution.

7.4.2.2.1   Demo

Here the detail for the demo size

HANA instance size

Standard_E4s_v3 with xx vCPU and yy GB memory

Accelerated networking

false

HANA disk configuration details.

`disks_type    =`	
`"Premium_LRS,Premium_LRS,Premium_LRS,Premium_LRS,Premium_LRS,Premium_LRS,Premium_LRS"`	
`disks_size     = "128,128,128,128,128,128,128"`	
`caching        = "None,None,None,None,None,None,None"`	
`writeaccelerator = "false,false,false,false,false,false,false"`	
`luns           = "0,1#2,3#4#5#6#7"`	
`names          = "data#log#shared#usrsap#backup"`	
`lv_sizes       = "100#100#100#100#100"`	
`paths          = "/hana/data#/hana/log#/hana/shared#/usr/sap#/hana/backup"`	

TABLE 7.2: NETWEAVER CONFIGURATION VARIABLES

netweaver_xscs_vm_size = "Standard_D2s_v3"
netweaver_app_vm_size = "Standard_D2s_v3"
netweaver_data_disk_type = "Premium_LRS"
netweaver_data_disk_size = 128
netweaver_data_disk_caching = ""ReadWrite""
netweaver_xscs_accelerated_networking = false
netweaver_app_accelerated_networking = false
netweaver_app_server_count = 2

7.4.2.2.2   Small

HANA instance size

Standard_E64s_v3 with xx vCPU and yy GB memory

Accelerated networking

true

HANA disk configuration details.

disks_type =	
"Premium_LRS,Premium_LRS,Premium_LRS,Premium_LRS,Premium_LRS,Premium_LRS"	
disks_size = "512,512,512,512,64,1024"	
caching = "ReadOnly,ReadOnly,ReadOnly,ReadOnly,ReadOnly,None"	
writeaccelerator = "false,false,false,false,false,false"	
luns = "0,1,2#3#4#5"	
names = "datalog#shared#usrsap#backup"	
lv_sizes = "70,100#100#100#100"	
paths = "/hana/data,/hana/log#/hana/shared#/usr/sap#/hana/backup"	

TABLE 7.3: **NETWEAVER CONFIGURATION DETAILS**

netweaver_xscs_vm_size = "Standard_D2s_v3"
netweaver_app_vm_size = "Standard_D2s_v3"
netweaver_data_disk_type = "Premium_LRS"
netweaver_data_disk_size = 128
netweaver_data_disk_caching = ""ReadWrite""
netweaver_xscs_accelerated_networking = false
netweaver_app_accelerated_networking = false
netweaver_app_server_count = 2

7.4.2.2.3 Medium

HANA instance size

Standard_M64s with xx vCPU and yy GB memory

Accelerated networking

true

HANA disk configuration details.

disks_type =	
"Premium_LRS,Premium_LRS,Premium_LRS,Premium_LRS,Premium_LRS,Premium_LRS,Premium_LRS,Premium_LRS,Premi	
disks_size = "512,512,512,512,512,512,1024,64,1024,1024"	
caching =	
"ReadOnly,ReadOnly,ReadOnly,ReadOnly,None,None,ReadOnly,ReadOnly,ReadOnly,ReadOnly"	
writeaccelerator = "false,false,false,false,false,false,false,false,false,false"	
luns = "0,1,2,3#4,5#6#7#8,9"	
names = "data#log#shared#usrsap#backup"	
lv_sizes = "100#100#100#100#100"	
paths = "/hana/data#/hana/log#/hana/shared#/usr/sap#/hana/backup"

TABLE 7.4: NETWEAVER CONFIGURATION DETAILS

netweaver_xscs_vm_size = "Standard_D2s_v3"
netweaver_app_vm_size = "Standard_E64s_v3"
netweaver_data_disk_type = "Premium_LRS"
netweaver_data_disk_size = 128
netweaver_data_disk_caching = "ReadWrite"
netweaver_xscs_accelerated_networking = false
netweaver_app_accelerated_networking = true
netweaver_app_server_count = 5

7.4.2.2.4   Large

HANA instance size

Standard_M128s with xx vCPU and yy GB memory

Accelerated networking

true

HANA disk configuration details.

disks_type =	
"Premium_LRS,Premium_LRS,Premium_LRS,Premium_LRS,Premium_LRS,Premium_LRS,Premium_LRS,Premium_LRS,Premi	

`disks_size`	`= "1024,1024,1024,512,512,1024,64,2048,2048"`
`caching`	`=`
`"ReadOnly,ReadOnly,ReadOnly,None,None,ReadOnly,ReadOnly,ReadOnly,ReadOnly"`	
`writeaccelerator = "false,false,false,true,true,false,false,false,false"`	
`luns`	`= "0,1,2#3,4#5#6#7,8"`
`names`	`= "data#log#shared#usrsap#backup"`
`lv_sizes`	`= "100#100#100#100#100"`
`paths`	`= "/hana/data#/hana/log#/hana/shared#/usr/sap#/hana/backup"`

TABLE 7.5: NETWEAVER CONFIGURATION DETAILS

netweaver_xscs_vm_size = "Standard_D2s_v3"
netweaver_app_vm_size = "Standard_E64s_v3"
netweaver_data_disk_type = "Premium_LRS"
netweaver_data_disk_size = 128
netweaver_data_disk_caching = "ReadWrite"
netweaver_xscs_accelerated_networking = false
netweaver_app_accelerated_networking = true
netweaver_app_server_count = 10

7.5 Salt Building Blocks

We have seen that resources are the most important elements in terraform, and there is an other resource type used as last step from the terraform process, the *Provisioner* resource.

It can be used to model specific actions on a remote machine in order to prepare them for other services.

The terraform *file provisioner* is used to copy directories *MAIN/*salt and *MAIN/*pillar from the machine executing Terraform to the newly created nodes.

As last step the terraform *remote-exec provisioner* is used, to call the script *provision.sh* on the remote node to run the salt provisioning steps. It comes from the terraform module *MAIN/ generic_modules/salt_provisioner/main.tf*.

So from this point on all work is done on the respective node itself. *

7.5.1 Our Architecture for the salt building blocks

Formulars: group of states give a context for building blocks e.g HANA States: combination of execution modules and other parts, have logic in and execute to a desired state Execution modules: basic execution modules, to provide the methods in the lower layer (shaptools) to salt shaptools: low level python wrapper (api) around sap utilities and commands

The provisioning workflow of the SAP building blocks consist of different steps:

1. Bootstrap salt installation and configuration

2. Execute OS setup operations. Register to SCC if needed, updated the packages etc, with help of executing the states within */srv/salt/os_setup*

3. Execute predeployment operations with help of execution of the */srv/salt/top.sls* states. It update hosts and hostnames, install the formular packages, etc

4. Execute deployment operations depending on the overall configuration settings like install SAP applications and configure and setup HA with the salt formulas.

7.5.2 Salt Overview

The SAP building block are created with help of SALT formulars after provisioning the virtual machines with terraform. The formulars are shipped as RPM packages with SUSE Linux Enterprise Server for SAP Applications

The salt formulas can be used with 2 different approaches: salt master/minion or only salt minion execution.

With the automation solution we use the salt minion option, the steps must be executed in all of the minions where the formulas are going to executed, which is done through a ssh connection.

The core of the Salt State system is the SLS, or **S**a**L**t **S**tate file. The SLS is a representation of the state in which a system should be in, and is set up to contain this data in a simple format.

There are 3 types of salt files used pillar files:: the *configuration* parameters where the data gets imported with help of jinja (map.jinja) and salt['pillar.get'] state files:: the *execution* definition in /srv/salt grains files:: *environment* parameters from the node itself and for handing over variables from terraform e.g. /etc/salt/grains

In Salt, the file which contains a mapping between groups of machines on a network and the configuration roles that should be applied to them is called a top file.

*

Top files are named *top.sls* by default and they are so-named because they always exist in the "top" of a directory hierarchy that contains state files and this directory hierarchy is called a state tree.

7.5.2.1   Salt pillar

Similar to the state tree, the pillar is comprised of .sls files and has a top file too. The default location /srv/pillar.

The pillar files define custom variables and data for a system.

When Salt pillar data is refreshed, each Salt minion is matched against the targets listed in the *top.sls* file. When a Salt minion matches a target, it receives all of the Salt pillar SLS files defined in the list underneath that target.

Directory structure for pillars.

```
/srv

├── pillar
│   ├── top.sls
│   ├── drbd
│   │   ├── cluster.sls
│   │   └── drbd.sls
│   ├── hana
│   │   ├── cluster.sls
│   │   └── hana.sls
│   ├── iscsi_srv.sls
│   └── netweaver
│       ├── cluster.sls
│       └── netweaver.sls
├── salt

...
```

The *top.sls* pillar file describes the needed pillar data for the respective role of the node.

State top.sls file.

```
base:

  'role:iscsi_srv':

    - match: grain

    - iscsi_srv
```

*

```
  'G@role:hana_node and G@ha_enabled:true':

    - match: compound

    - hana.cluster


  'role:drbd_node':

    - match: grain

    - drbd.drbd

    - drbd.cluster


  'role:netweaver_node':

    - match: grain

    - netweaver.netweaver


  'G@role:netweaver_node and G@ha_enabled:true and P@hostname:.*(01|02)':

    - match: compound

    - netweaver.cluster
```

To run an initial deployment without specific customization, you can use pillar files stored in the
_MAIN/pillar_example/automatic` folder, as these files are customized with parameters coming
from terraform execution. The pillar files stored there are able to deploy a basic functional set
of clusters in all of the available cloud providers.

To adapt the deployment to your scenario, you should provide your own pillar data files and
there are some basic examples within the directory *MAIN/pillar_example*. As the pillar files pro-
vide data for the salt-formulas, you can find all of the pillar possible options in each formula
project.

❗ Important

Pillar files are expected to contain private data such as passwords required for the auto-
mated installation or other operations. Therefore, such pillar data need to be stored in an
encrypted state, which can be decrypted during pillar compilation.

SaltStack GPG renderer provides a secure encryption/decryption of pillar data. The con-
figuration of GPG keys and procedure for pillar encryption are described in the Saltstack
documentation guide:

*

1. SaltStack pillar encryption (https://docs.saltstack.com/en/latest/topics/pillar/#pillar-en-
   cryption) ↗

2. SaltStack GPG RENDERERS (https://docs.saltstack.com/en/latest/ref/renderers/all/

This is not done by the project and you need take care of this by yourself

7.5.2.2 Salt States

Salt state files are organized into a directory tree, called the Salt state tree, in the */srv/salt/* directory.

Directory structure for Salt state files.

```
/srv
├── pillar
....
├── salt
│   ├── cluster_node
│   │   ├──
│   ├── default
│   │   ├──
│   ├── drbd_node
│   │   ├──
│   ├── hana_node
│   │   ├──
│   ├── iscsi_srv
│   │   ├──
│   ├── _modules
│   │   ├──
│   ├── monitoring_srv
│   │   ├──
│   ├── netweaver_node
│   │   ├──
│   ├── os_setup
│   │   ├──
│   ├── provision.sh
│   ├── qa_mode
│   │   ├──
│   ├── sshkeys
```

*

The *top.sls* file describes two environments for the nodes, *pre-deployment* and *base* which reflect the steps 3 and 4 of the workflow above. For each role of the nodes there more detailed files responsible.

The pre-deployment is needed, as we can not install formulas and use them directly in the same execution.

State top.sls file.

```
predeployment:
  'role:hana_node':
    - match: grain
    - default
    - cluster_node
    - hana_node

  'role:netweaver_node':
    - match: grain
    - default
    - cluster_node
    - netweaver_node

  'role:drbd_node':
    - match: grain
    - default
    - cluster_node
    - drbd_node

  'role:iscsi_srv':
    - match: grain
    - iscsi_srv

  'role:monitoring_srv':
    - match: grain
    - default
    - monitoring_srv
```

```
  'role:drbd_node':

    - match: grain

    - drbd

    - cluster


  'role:netweaver_node':

    - match: grain

    - netweaver


  'G@role:netweaver_node and G@ha_enabled:true and P@hostname:.*(01|02)':

    - match: compound

    - cluster
```

7.5.2.3 Salt grains

SaltStack comes with an interface to derive information about the underlying system. This is called the *grains* interface, because it presents salt with grains of information. It collects static informations about the underlying managed system, like the operating system, domain name, IP address, kernel, OS type, memory, and many other system properties.

We use custom grains to match the roles and the further states.

The *role* is a *custom grains* define with help of the terraform file *salt_provisioner.tf* for the respective building block.

CAUTION:

```
If you use the salt formulars independent from the terraform templates, you need to
 take care of providing all needed variables by yourself which normally get set by the
 _salt_provisioner.tf_.
```

7.5.2.4 State details

If you target a directory during a *state.apply* or in the state Top file, salt looks for an init.sls file in that directory and applies it.

Within the *os_setup* there is e.g. the minion configuration file

```
|   ├── os_setup
|   |   ├── init.sls
|   |   ├── ip_workaround.sls
|   |   ├── minion_configuration.sls
|   |   ├── packages.sls
```

```
|   |   ├── registration.sls
|   |   └── repos.sls
```

One interesting file is the *minion_configuration.sls* as it provides the configuration how and where salt / the minion looks for salt states and salt formulas.

If we look deeper into one of the directories, e.g. *hana-node* we will find more files in these directories.

HANA Node state files.

	hana_node
		├── download_hana_inst.sls
		├── files
			└── sshkeys
			├── cluster.id_rsa
			└── cluster.id_rsa.pub
		├── hana_inst_media.sls
		├── hana_packages.sls
		├── **init.sls**
		└── mount
		├── azure.sls
		├── gcp.sls
		├── **init.sls**
		├── mount.sls
		├── mount_uuid.sls
		└── packages.sls

Salt executes what is in *init.sls* in the order listed in the file. When an salt file is named init.sls it inherits the name of the directory path that contains it. This formula/state can then be referenced with the name of the directory.

In our case here, it first it gets the SAP HANA Media with help of *hana_ins_media,* create the mountpoints and partition disks for SAP HANA and enter them into the fstab with help of the states in the *mount* directory. Similar as before, the starting point is again the *init.sls* file.

After all is processed within *mount,* it gets back to the file *hana_packages,* which then install the RPM packages *shaptools* and *saphanabootstrap-formula* which get shipped with SUSE Linux Enterprise Server for SAP Applications.

All other states files get processed in the same way as the example above.

7.5.3   Salt formula packages

Formulas are pre-written Salt States. They are as open-ended as Salt States themselves and can be used for tasks such as installing a package, configuring, and starting a service, setting up users or permissions, and many other common tasks. Each Formula is intended to be immediately usable with sane defaults without any additional configuration.

Our formulas are configurable by including data in Pillar, what we discussed above. During RPM install, the the files of the packages end up in the directory */usr/share/salt-formulas/states,* which we had defined as directory where salt searches for file in addition to /srv/salt (see os_setup state above).

shaptools package. If you have wondered above about the directories *modules_ and* states_, they come from the install of the package shaptools and provide a python wrapper for sap command line tools as API, in order to make it simple to be used from salt. This package is a base dependency for most of our formula packages as it provides the SAP commands.

```
|   ├── _modules
|   |   ├── ...
|   ├── _states
|   |   ├── ...
```

7.5.3.1   HANA formula

The main work of preparing the node for HANA and installing HANA is done by the *saphana-bootstrap-formula.*

The structure is similar what you have seen above for pillars and states but lives in the directory */usr/share/salt-formulas/states/…*

```
states/
└── hana
    ├── defaults.yaml
    ├── enable_cost_optimized.sls
    ├── enable_primary.sls
    ├── enable_secondary.sls
    ├── exporter.sls
    ├── init.sls
    ├── install.sls
    ├── map.jinja
    ├── packages.sls
    ├── pre_validation.sls
    └── templates
```

*

```
            ├── hanadb_exporter.j2
            ├── scale_up_resources.j2
            └── srTakeover_hook.j2
```

Salt includes the Jinja2 template engine which can be used in Salt state files, Salt pillar files, and other files managed by Salt.

Salt lets you use Jinja to access minion configuration values, grains and Salt pillar data, and call Salt execution modules. One of the most common uses of Jinja is to insert conditional statements into Salt pillar files.

1. The formula package is installed through the HANA Node state files

2. If you want to install it manual, please use zypper, as it will include the other dependent packages like salt-shaptools and habootstrap-formula

```
zypper install saphanabootstrap-formula
```

The salt formula need some input data through a pillar file, which is coming from the main project file (MAIN/pillar/... or on the node /srv/pillar), or if you use it standalone it need to be provided by you.

Example HANA pillar.

```
hana:

  saptune_solution: 'HANA'

  nodes:

    - host: 'hana01'

      sid: 'prd'

      instance: "00"

      password: 'SET YOUR PASSWORD'

      install:

        software_path: '/sapmedia/HANA'

        root_user: 'root'

        root_password: ''

        system_user_password: 'SET YOUR PASSWORD'

        sapadm_password: 'SET YOUR PASSWORD'

      primary:

        name: PRIMARY_SITE_NAME

        backup:

          key_name: 'backupkey'

          database: 'SYSTEMDB'

          file: 'backup'

        userkey:
```

```
        user_password: 'SET YOUR PASSWORD'

        database: 'SYSTEMDB'


    - host: 'hana02'

      sid: 'prd'

      instance: "00"

      password: 'SET YOUR PASSWORD'

      install:

        software_path: '/sapmedia/HANA'

        root_user: 'root'

        root_password: ''

        system_user_password: 'SET YOUR PASSWORD'

        sapadm_password: 'SET YOUR PASSWORD'

      secondary:

        name: SECONDARY_SITE_NAME

        remote_host: 'hana01'

        remote_instance: "00"

        replication_mode: 'sync'

        operation_mode: 'logreplay'

        primary_timeout: 3000
```

1. The formular is executed within the salt of *hana_node* state files

2. If you want to execute the formular manually, salt

```
salt '*' state.apply hana_node.sls
```

So with help of the pillar data and the state file and the formular, salt will create all needed configuration on the node, installs HANA and if enabled install hana systemreplication and set up the pacemaker cluster, right for Azure.

The *templates* directory provides the needed files for cluster rules, the needed hook for HANA and the monitoring exporter. All the values come from the best practice guides SUSE created with the Cloudprovider Azure for the HA scenario.

*

7.5.3.2 Netweaver formula

The SAP Netweaver deployment is performed using the *sapnwbootstrap-formula* and uses as of today only SAP HANA as a database.

The formula has some hard dependencies and all of them must be in place for a successful netweaver deployment. In order to deploy a correct Netweaver environment a NFS share is needed (SAP stores some shared files there). The NFS share must have the folders *sapmnt* and *usrsapsys* in the exposed folder. The folders are created with the Netweaver SID name (for example /sapdata/HA1/sapmnt and /sapdata/HA1/usrsapsys). This subfolders content is removed by default during the deployment.

Second, the SAP installation software (swpm) must be available in the system. To install the whole Netweaver environment with all the 4 components, the swpm folder, sapexe folder, Netweaver Export folder and HANA HDB Client folders must already exist, or be previously mounted when provided by external service, like NFS share. The netweaver.sls pillar file must be updated with all this information. Netweaver Export and HANA HDB Client folders must go in additional_dvds list.

The structure is similar what you have seen above for the HANA formula.

```
states/
└── ...
    └── netweaver
        ├── defaults.yaml
        ├── ensa_version_detection.sls
        ├── extract_nw_archives.sls
        ├── ha_cluster.sls
        ├── init.sls
        ├── install_aas.sls
        ├── install_ascs.sls
        ├── install_db.sls
        ├── install_ers.sls
        ├── install_pas.sls
        ├── install_pydbapi.sls
        ├── map.jinja
        ├── monitoring.sls
        ├── pillar.example
        ├── pre_validation.sls
        ├── saptune.sls
        ├── setup
        │   ├── init.sls
        │   ├── keepalive.sls
        │   ├── mount.sls
        │   ├── packages.sls
        │   ├── sap_nfs.sls
        │   ├── shared_disk.sls
        │   ├── swap_space.sls
        │   ├── users.sls
```

```
        │   └── virtual_addresses.sls
        └── templates
            ├── aas.inifile.params.j2
            ├── ascs.inifile.params.j2
            ├── cluster_resources.j2
            ├── db.inifile.params.j2
            ├── ers.inifile.params.j2
            └── pas.inifile.params.j2
```

As you know from earlier descriptions, we need a pillar file with the configuration. There is one example in the path which could be used as base for a standalone salt usage. In general the pillar data get handed over from the terraform main project.

As SAP Netweaver has in an HA environment more nodes, therefore the pillar file is much bigger as eg. the one for HANA. So please have a look by yourself of the example file.

Similar as before, the starting point is the *init.sls* file where the workflow is defined.

The *templates* directory provides the needed files for NW cluster rules and the values come from the best practice guides SUSE created with the Cloudprovider Azure for the ERS scenario.

In addition here are the templates which are used by SWPM for a automated handfree installation of the SAP Netweaver services.

7.5.4   High Availability formula

The *habootstrap-formula* provide the needed cluster setups for SAP HANA, SAP Netweaver, or if needed for the HA NFS service build with drbd. It will take care of

The formula will be, similar to all the other formulas used, installed in /usr/share/salt-formulas/states/cluster.

```
states
├── cluster
│   ├── create.sls
│   ├── defaults.yaml
│   ├── init.sls
│   ├── join.sls
│   ├── map.jinja
│   ├── monitoring.sls
│   ├── ntp.sls
│   ├── packages.sls
│   ├── pre_validation.sls
│   ├── remove.sls
│   ├── resource_agents.sls
│   ├── sshkeys.sls
```

*

```
|     ├── support
|     |   └── ssh_askpass
|     └── watchdog.sl
```

The main difference to the HANA and Netweaver formula is that the *init.sls* make already use of *jinja*. Jinja is the default templating language in SLS files and get evaluated before YAML, which means it is evaluated before the States are run.

The most basic usage of Jinja in state files is using control structures to wrap conditional or redundant state elements

7.5.5 Additional Services

The additinal services depend on what is used or available of the cloudprovider, but needed by SAP HANA or SAP Netweaver or the HA services.

7.5.5.1 NFS service

To build an HA-NFS service, we use the above describe *habootstrap-formula* together with *drbd-formula* to mirror the data between two nodes and the _linux nfs-server: packages been setup with the saltstack _nfs_formula (see https://github.com/saltstack-formulas/nfs-formula ↗)

DRBD®– software is a distributed replicated storage system for the Linux platform. It is implemented as a kernel driver, several userspace management applications, and some shell scripts. So simplified, think about it as an raid-1 over network.

Details are available at the SUSE documentation page for the SLE HA Extension https://documentation.suse.com/sle-ha/15-SP2/single-html/SLE-HA-nfs-quick/#art-sleha-nfs-quick ↗

7.5.5.2 Fencing service

If the setup is using HA for SAP Netweaver or SAP HANA or with the NFS service, and there is mechanism for fencing of the virtual machines over an API we use the SUSE method of using a SBD-device. Such a SBD-Device is normally a raw shared disk beween two nodes.

Unfortunately not all clouds are able to provide a raw shared disk, but with the help of linux native services (iSCSI) we can build this by our own.

We use here the *iscsi-formula* provided by saltstack itself, see https://github.com/saltstack-formulas/iscsi-formula ↗ to provide the nodes of the cluster a raw-shared-disk with help of a *iscsi target* for the SBD fencing mechanism.

*

It gets configured through the pillar files we provided through the role *iscsi_srv*

The use of possible fenching method depends on the cloud providers possibilities. As of today SBD is needed only for Azure, but it is a general method which could be used nearly independent of the base infrastructure.

*

8 Migration

FixMe - Varius sit amet mattis vulputate. Nisi scelerisque eu ultrices vitae auctor eu augue ut. Integer vitae justo eget magna fermentum iaculis eu non diam. Rhoncus urna neque viverra justo. Elementum tempus egestas sed sed risus. Porta nibh venenatis cras sed felis eget velit aliquet sagittis. Venenatis a condimentum vitae sapien pellentesque. Magna ac placerat vestibulum lectus mauris ultrices eros in cursus. Nibh cras pulvinar mattis nunc. Tempor orci dapibus ultrices in iaculis nunc. Sapien nec sagittis aliquam malesuada bibendum arcu vitae elementum. Nisi porta lorem mollis aliquam. Laoreet id donec ultrices tincidunt arcu non sodales.

FIGURE 8.1: SOLUTION ARCHITECTURE - FIXME MIGRATION

*

9 Summary

More and more companies move to a computing *as a service,* rather than *as a product,* which bring new possibilities for innovations, but reshaping the landscape, will bring new challenges where SUSE can help with to solve them.

The SUSE solution manages complex operations with automation and help ease the transition to Linux and the cloud, and reduce the problem resolution time with insights to the SAP infrastructure landscape.

It help to deliver SAP services faster, more efficiently and with less risk.

*

Glossary

- FixMe - Deployment Type(s)::

- Factor(s)

- Flavor(s)
  +

Python

A scripting language. It interacts with lower-layer utilities such as crm shell and several SAP commands, including SAP HANA management tools.

Salt (also SaltStack)

A configuration infrastructure management system written in Python. Due to its modular approach, it is often referred as SaltStack. Salt has as a client/server architecture. The server (also called the Salt Master) acts as a central control unit for the Salt clients. The other supported setup option is called masterless.

Salt Grains

Static data about Salt clients. Grains contain information about the operating system that is running, the CPU architecture in use, and much more. Grains can also be set to assign values to Salt clients.

Salt Formulas

Formulas are pre-written Salt States.
For more information about Salt, refer to the upstream documentation at https://docs.saltstack.com ↗ .

Salt Master

Manages the infrastructure and the Salt clients within it. It can execute commands remotely on Salt clients and manage their state. The Salt Master captures grains sent from Salt clients and decide what to do with this information.

Salt Client (sometimes Minion)

A server or machine often controlled by the Salt Master. Its main purpose is to execute commands sent from the Salt Master, report data back, and send information about itself.

Salt State

*

YAML text file to maintain consistency across your environment. Salt states can be executed.

Terraform

An "infrastructure as code" software tool. It deploys the required infrastructure in cloud or virtual environments and AutoYaST for on-premises deployments.

*

A   Appendix

Draft

*