



SUSE SAP automation solution for Azure



SUSE SAP automation solution for Azure

Publication Date: 2021-02-16

SUSE LLC

1800 South Novell Place

Provo, UT 84606

USA

<https://documentation.suse.com> 

Contents

v

I	SUSE SAP AUTOMATION SOLUTION FOR AZURE	1
1	Preface	2
2	Introduction	3
3	Strategy	5
3.1	Context	5
4	Business	7
4.1	SUSE SAP Automation Coverage	8
4.2	Prerequisites for SAP workloads in public clouds	9
5	Application	10
5.1	SUSE Linux Enterprise Server for SAP Applications	10
5.2	SAP Application	10
5.3	Presenting the SAP Media	10
5.4	Terraform	10
5.5	SALT	12
	Overview of the available formulas which are used within the SUSE Automation framework.	13
5.6	Monitoring	15
	SAP HANA Database Exporter	15
	• High Availability Cluster Exporter	16
	• SAP Host Exporter	16

6 Technology 17

- 6.1 Terraform 17
- 6.2 Salt 17
- 6.3 SAP Sizing 18
- 6.4 Building Blocks 18
- 6.5 High Availability 18
- 6.6 Additional Services 19
 - NFS service 19 • Fencing service 20 • Monitoring service 22

7 Physical 23

- 7.1 Prerequisites 23
- 7.2 Get the project 24
- 7.3 Terraform Building Blocks 25
- 7.4 Simple Install 27
 - Terraform file details 30 • SAP Sizing 33
- 7.5 Salt Building Blocks 41
 - Our Architecture for the Salt building blocks 41 • Salt Overview 42 • Salt formula packages 48 • High Availability formula 53 • Additional Services 53

8 Summary 55

A Appendix 57

Authors

Peter Schinagl,	SUSE Software Solutions Germany GmbH,	peters@suse.com
Stephen Mogg,	SUSE Software Solutions UK LTD,	Stephen.Mogg@suse.com
Abdelrahman Mohamed,	SUSE Software Solutions Germany GmbH,	abdelrahman.mohamed@suse.com
Bryan Gartner,	SUSE Software Solutions LLC,	bryan.gartner@suse.com

I SUSE SAP automation solution for Azure

1	Preface	2
2	Introduction	3
3	Strategy	5
4	Business	7
5	Application	10
6	Technology	17
7	Physical	23
8	Summary	55
	Glossary	56
A	Appendix	57

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 Terraform, 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

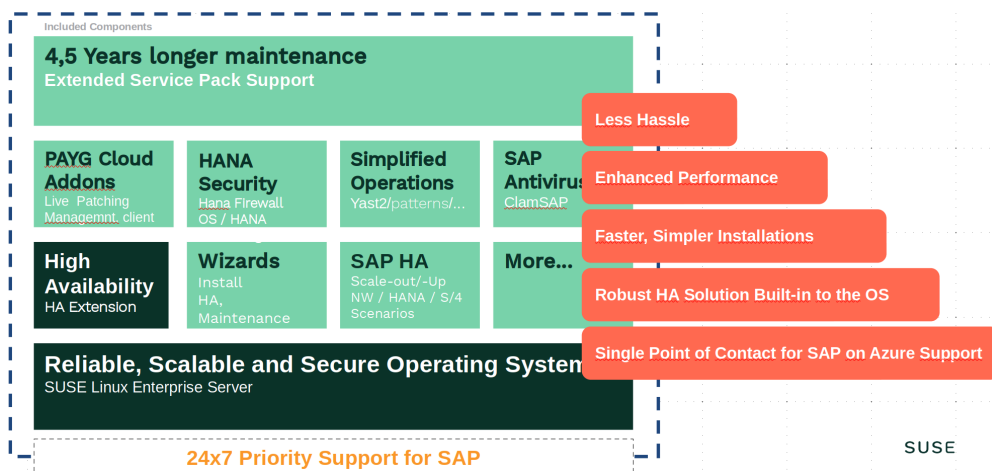
¹ link: [Archimate Enterprise Architecture \(https://pubs.opengroup.org/architecture/archimate3-doc\)](https://pubs.opengroup.org/architecture/archimate3-doc) 

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 development, test, and production.

Modernize

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

Accelerate

customer deployments and migrations to the public cloud.

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

Highly available infrastructure configurations raise additional complexities that further delay time to value.

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 the environment after the installation is complete.

The major motivation is to improve, simplify, and unify the installation of the SAP Landscape on SUSE Linux Enterprise Server for SAP Applications, clearly standardize deployments, and allow customers to use one level of tooling in various ways – through a Command Line interface, through some GUI driven processes, and through SUSE Manager or other automation framework. To achieve this, SUSE has adopted a more modern approach, infrastructure-as-code, which helps customers reduce effort and errors during deployments.

In recent years, SUSE Linux Enterprise Server and many other SUSE products ship 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 application 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 and error-free setup of an SAP Landscape. Once the deployment is complete, administrators can log in to start customizing the SAP System.

As part of the deployment, SUSE added the ability provide insights into the 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 set up a monitoring environment for the SAP HANA and SAP NetWeaver clusters.

3 Strategy

Most SAP services are deployed on-premises with well-established procedures, but significant planning must go into these deployments. For example, workload growth must be estimated and planned for in hardware requirements, often several years in advance.

Predicting the future is not easy. When considering requirements, such as storage capacity, many factors may affect the system over its lifespan. Selections often end up being just "best guesses."

With today's quickly changing environments, many businesses need to accelerate innovation and increase agility across their entire landscape in order to achieve a faster time-to-market and manage costs. Migrating products and services to the cloud can help businesses become more flexible and agile to meet changing business demands.

One key benefit of the cloud is that a business no longer needs to plan hardware sizing for the next five or more years. Larger, faster, or even smaller infrastructure is only one reboot away.

However, "rightsizing" (or infrastructure optimization) remains an important consideration. Businesses that actively manage "rightsizing" their environment can cut operating costs by 30 to 60 percent.

3.1 Context

There are many benefits gained when moving SAP workloads to the cloud:

Quick deployment

If you need fast application implementation and deployment, the cloud is the best choice. You can set up a cloud environment within a few hours, whereas, in-house IT infrastructure can take days or even months to order, install, configure, and bring online.

With SUSE's automation solution for SAP, IT teams can easily and quickly implement and deploy their SAP Landscape remotely in the cloud.

Reduce Costs

Many businesses experience large CapEx (Capital Expenditure) to maintain IT infrastructure, with purchases often in advance of actual need. The cloud helps businesses transform to a more efficient OpEx (Operating Expenditure) model, with several consumption options, including "pay as you use," that let businesses manage their infrastructure costs.

SUSE, together with the cloud providers, can offer the right options to control costs, but this also requires businesses to adapt how they use the resources and SAP software.

Scalability & Flexibility

With the cloud, businesses can scale up or scale down resources as needed. This makes it much easier to "rightsize" the environment and ensure efficient use of resources that can also adjust to meet changing business demands.

Maintenance

With the cloud, IT departments no longer have to worry about managing and maintaining the hardware and underlying infrastructure. The cloud service provider handles this, freeing up company resources to focus on innovation and other business needs.

Businesses who deploy their SAP Landscapes with SUSE products and SUSE best practices automation experience simplified maintenance and less downtime.

Resiliency

Uptime is of prime importance to ensure day-to-day business operations run smoothly. Moving to the cloud maximizes uptime and reduces downtime. The cloud improves disaster recovery and business continuity without the need to spend a huge amount of capital on robust disaster recovery tools. And, cloud providers offer a variety of services to help protect businesses from security threats and outages.

SUSE's SAP HA automation augments these services to further reduce downtime of SAP applications.

Remote access

The cloud allows employees to access data from anywhere and at any time, making business more flexible and increasing productivity.

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

Overall, SUSE and public clouds offer significant benefits for all customers, regardless of size. The use of cloud resources can lower infrastructure costs and improve the scalability, agility, flexibility, and availability of SAP applications.

4 Business

This 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 responsibility (e.g., Networking, DB, OS, Application), as most operations can be performed from a central control plane. However, this should not mean that this specialized knowledge is no longer needed. Functional teams still exist and will need to work together, this is often best achieved with a DevOps approach utilizing Infrastructure-as-Code.

When implementing SAP in the cloud, knowledge is required of the cloud infrastructure and the various possibilities this affords along with a good understanding of the operating system and the tooling surrounding it; e.g., High Availability (HA). Finally, an understanding around planning for the application usage and sizing is needed.

SAP architectures need to be fine-tuned based on customer requirements around system availability (i.e., 99.99%, 99.95% or 99.9%). Each Single Point of Failures (SPOF) in the components and services will need to be identified and protected against. This is normally achieved with an HA Cluster. And, other SPOFs within the infrastructure will need to be protected against with some form of redundancy.

If you look at a typical SAP implementation you will find:

1. SAP Central Services (ASCS/ERS)
2. a Database (e.g., SAP HANA)
3. a Primary Application Server (PAS)
4. shared storage (NFS)

In the above list, items 1, 2, and 4 are potential SPOFs.

SUSE's SAP Automation will try to eliminate all of these SPOFs by providing HA cluster implementations to ensure automated failover, data protection, and higher system availability.

SAP Central Services (ASCS/ERS)

You need at least 2 nodes to configure an ASCS/ERS HA 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 nodes to configure SAP HANA HA/DR cluster in a scale-up deployment. The SUSE HA Extension is used to detect system failures and facilitate automatic failover.

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

This is one of the main benefits of the SUSE SAP Automation project: all the required infrastructure and configuration can be created in order to maximize the SAP System availability.

4.1 SUSE SAP Automation Coverage

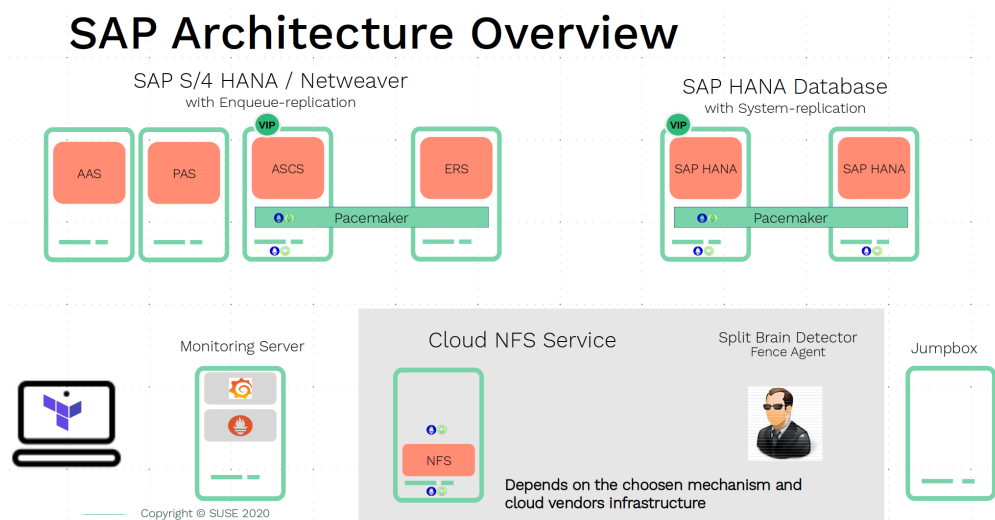
SAP HANA and Netweaver applications can be deployed in many different scenarios and combinations between them. The automation is constructed from 'building blocks' which are modular and reusable and can be used to deploy a single install through to full cluster deployment.

The following scenarios are supported:

- HANA single node
- HANA HA Scale-up System Replication
 - Performance Optimized Scenario
 - active/passive
 - active/readonly
 - Cost Optimized Scenario
- Netweaver
- Netweaver HA with Enqueue Replication Version (ENSA1)
- S/4 HANA

SUSE Engineering continues to develop new scenarios based on the demands of customers and partners.

The overall deployment using SUSE SAP Automation looks as follows:



4.2 Prerequisites for SAP workloads in public clouds

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


1. License for the SAP software to be deployed is required.
2. Ensure understanding of the resource requirements of the SAP workloads (via an SAP System Sizing exercise).
3. Use certified instance types based on the capacity required by SAP software.
4. Ensure suitable network connectivity is provided (bandwidth, latency, and packet loss) within the cloud environment for SAP workloads.
5. Deploy only certified operating systems on which the SAP workloads will run.
6. Have a good operational knowledge of the Linux OS, SAP systems operations, and the cloud infrastructure.
7. Where highly available solutions are deployed, a deep understanding of the HA concepts, tooling, and how this functions within and alongside the resiliency capabilities of the cloud infrastructure are essential.

5 Application

5.1 SUSE Linux Enterprise Server for SAP Applications

SUSE Linux Enterprise Server for SAP Applications is a product formed from a bundle of software and services. It is targeted specifically at customers running SAP workloads. At its foundation is SUSE Linux Enterprise Server and the High Availability Extension with many additional components and benefits for running SAP Applications.

5.2 SAP Application

In order to use the automation project, there are preliminary steps which need to be taken. One of these is to prepare the SAP installation software. The SAP software can be downloaded from <https://launchpad.support.sap.com/#/softwarecenter> . This will need to be performed manually before starting the automated deployment.

5.3 Presenting the SAP Media

After downloading the required SAP software, the files must be presented via cloud storage so it is accessible from the new installed virtual machines / instances.

Azure offers shared storage (Azure Files) for applications using the Server Message Block (SMB) protocol, providing a simple way to upload the SAP media and use it from the installed machines for the SAP installation.

To use Azure Storage, start by creating 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 an 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 infrastructure components.

Terraform is seen as cloud-agnostic and allows a single configuration to be used to manage multiple providers. This simplifies management and orchestration, helping operators build large-scale multi-cloud infrastructures.

It is important to note that Terraform can not simply create a landscape in another cloud with the press of a button. The cloud providers use different types of infrastructure. For example the VMs, load balancers, and other services offered by AWS are very different to those in Azure and Google Cloud.

Terraform's approach is that code is written specific to a (cloud) provider and will take advantage of the provider's unique functionality. While the code needs to be modified when used on a different cloud provider, being able to use the same language and toolset for all providers makes this effortless.

The name Terraform uses for these cloud specific 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 deploy in order to support the application. One of the first steps is to run the `terraform` command, this will generate an execution plan that describes the actions Terraform will perform to get to the planned desired state. The plan is in the form of a list of cloud infrastructure to create, delete, and modify. If this looks correct, the final step is to execute the plan to create the described infrastructure.

SUSE provides Terraform configuration files for AWS, Azure, Google Cloud and libvirt.

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

In addition, Azure provides an easy-to-access, web-based command line (Cloud Shell), where Terraform is already pre-installed.

<https://shell.azure.com> 

You will find documentation for it at <https://docs.microsoft.com/en-us/azure/cloud-shell/overview> 

As Azure provides different types of storage suitable for supporting SAP workloads, it is important to fully understand the SAP storage requirements for Azure.

The suggestions from SUSE's Terraform files for the storage configurations are meant as a good starting point.

You should still analyze the storage utilization patterns during runtime of the application. You could realize, for example, that you are not utilizing all the storage bandwidth or IOPS provided, and you might consider downsizing storage. Alternatively, you could see that your workload needs more storage throughput than suggested; in which case, you might choose to change the capacity, IOPS, or throughput to optimize the configuration. Azure offers different storage types to allow you to find and select the right storage to support your SAP workload and meet your capacity, latency, throughput, IOPS, and cost needs.

5.5 SALT

SaltStack's configuration management system lets you define the applications, files, and other settings required on a specific system. The running system is continuously evaluated against the defined configuration, and changes are made as necessary.

- Salt works with "States" which express the required 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 used to build the system.

SLES-for-SAP Applications ships with all the Salt tools as part of the distribution and are available to use as needed.

Salt formulas can be applied in two ways:

Salt Master with Salt Minion

All steps are initiated on the Salt master, a central management system, which sends instructions to Salt minions, running on managed systems, to perform the required configuration actions.

Salt Minion Only

All steps defined in Salt formulas are executed on the individual systems by standalone Salt minions. This is the approach used by the SUSE Automation framework as it removes the need for a central master system.

5.5.1 Overview of the available formulas which are used within the SUSE Automation framework.

5.5.1.1 Netweaver

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

- Extract the required SAP files for SAP Media (.tar,.sar,.exe)
- Set up:
 - ASCS instance
 - ERS instance
 - PAS instance
 - AAS instance
 - Database instance (currently only HANA)

Besides that, the formula sets up all of the prerequisites, such as:

- Hostnames
- Virtual addresses
- NFS mounts
- Shared disks
- SWAP partitions

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

5.5.1.2 HANA

The HANA bootstrap formula takes care of the following:

- Extract the required SAP files for SAP Media (.tar,.sar,.exe)
- Install SAP HANA

- Apply "saptune" for HANA to configure and tune the OS for HANA usage
- Configure SAP System Replication
- Preconfigure the High Availability cluster requirements
- Configure the SAP HANA Prometheus exporter

5.5.1.3 HA

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

- Create and configure the High Availability cluster, pacemaker, corosync, Fencing, and SAP resource agents
- Adjustments for the Azure Infrastructure
- SBD for fencing
- Handle Netweaver, HANA and DRBD

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 prerequisites (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 fencing (agent or SBD)
- Configure Corosync
- Configure the resource agents
- Install and configure the monitoring *ha_cluster_exporter*

Depending on the fencing requirements it may need an iSCSI server to provide a raw shared disk for the fencing with SBD, where we use the *iscsi-formula* from SaltStack.

5.5.1.3.1 Other dependent services

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 bootstrap formula
- HA bootstrap formula
- NFS formula from SaltStack to install and configure nfs server and client

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.

5.6 Monitoring

SUSE continually works to improve user experience. One of the developments is how to provide a modern solution to monitor the several High Availability clusters that manage SAP HANA and SAP Netweaver. The Monitoring components use the Prometheus toolkit and the Grafana project to visualize the data. In order to be able to monitor the clusters on either HANA or Netweaver, SUSE has written Prometheus exporters which ship as part of SLES for SAP.

5.6.1 SAP HANA Database Exporter

The exporter provides metrics from more than one database or tenant. Specifically, 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 gives administrators the ability to easily monitor cluster status and health. The following capabilities are included:

- 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 showing:

- SAP start service process list
- SAP enqueue server metrics
- SAP application server dispatcher metrics
- SAP internal alerts



Tip

The gathered metrics are the data that can be obtained by running the sapcontrol command.

6 Technology

6.1 Terraform

What is Terraform?

Terraform along with the specific cloud provider modules is used to create the infrastructure to support the SAP application and supporting services.

The Terraform templates describe everything needed to create the desired infrastructure components. They also provide a range of pre-defined settings to simplify creation of the correct 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*.

An SAP landscape is made up of groups of machines, each machine in the group performing a role. These groups of machines work in concert with each other to create an application stack. To effectively manage these groups of machines, an administrator needs to be able to create roles for those groups. As an 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 is used to reference the building blocks for the SAP Landscape.

6.3 SAP Sizing

To make the SAP sizing simpler, SUSE has introduced pre-defined sizes with well-known abbreviations from T-Shirt sizes: Small (S), Medium (M), and Large (L).

The Small (S) size is targeted at non-production scenarios, whereas Medium (M) and Large (L) sizes are recommended for production setups and certified instance types should be used.

It is possible to tweak these pre-defined sizes or create a set of custom settings.

Sizing is critical. It includes choosing the right, SAP-certified instance types from the cloud provider, the right number of disks to support I/O requirements, and the right network options to meet throughput needs.

6.4 Building Blocks

The main building blocks for an SAP Landscape are the *Application Layer*, based on Netweaver with SAP Central Services (xSCS), a Primary Application Server (PAS), and Additional Application Server (AAS), as well as the *Database Layer*, which, for this context, is SAP HANA.

There are two possible models for how SAP Business Suite can be deployed: a centralized deployment where everything runs on one server or a distributed deployment where every service has its own node.

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

The distributed deployment is the recommended way for production environments where each of the SAP application layer components is independently installed on different instances. This is the scenario used within the automation project.

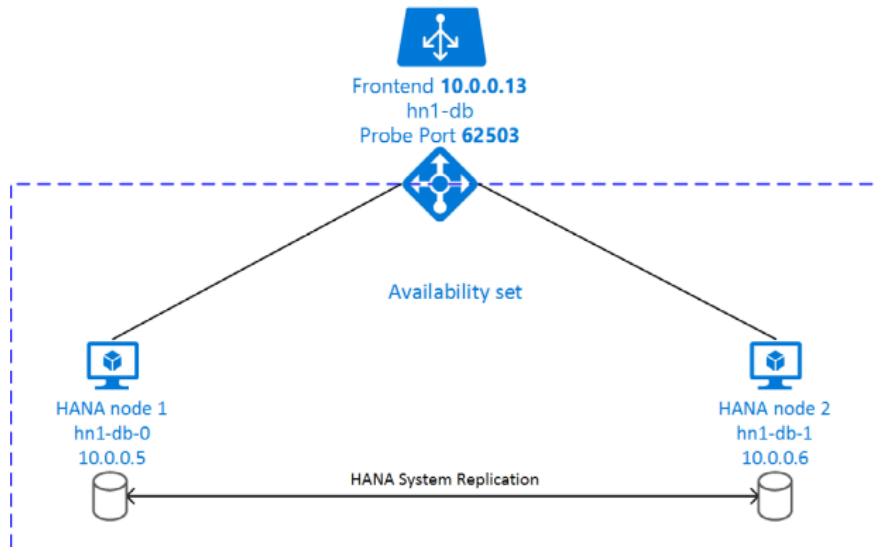
6.5 High Availability

There are two main parts in an SAP system which should be made highly available in order to achieve less downtime and eliminate single points of failure within the SAP Landscape.

For the Central Services this is Enqueue Replication, and for the database it is the HANA System Replication.

For each of these building blocks, one additional machine is required to build a two-node cluster within 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.

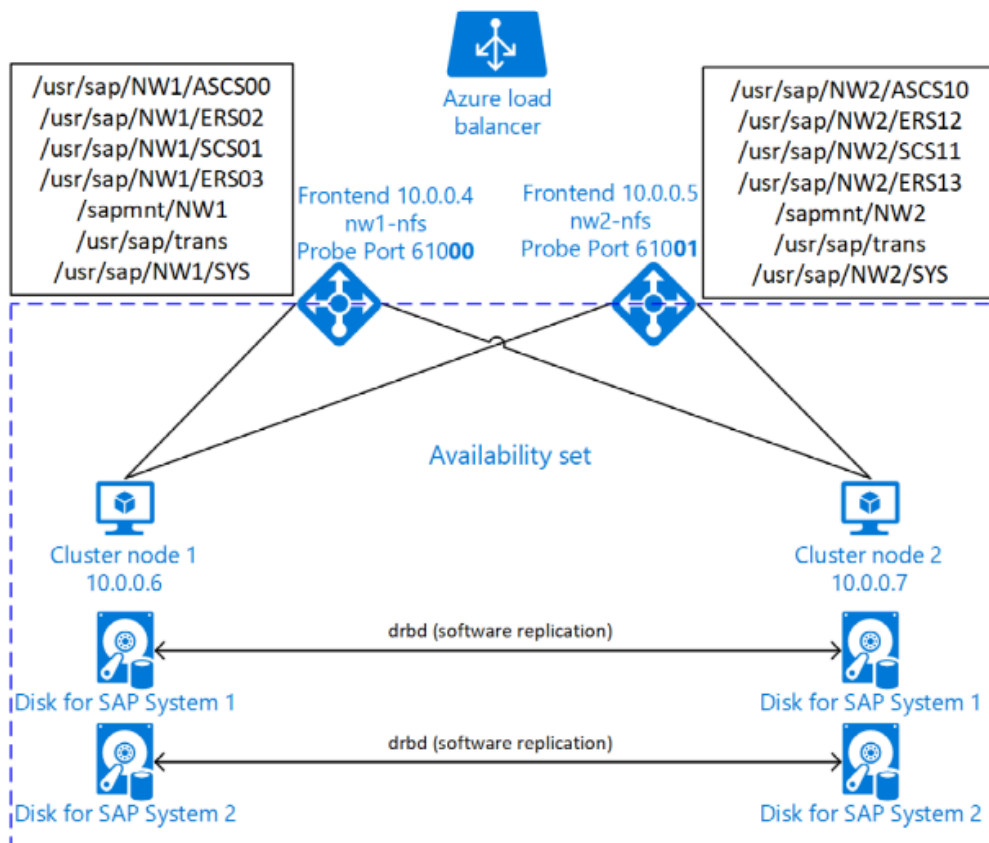


6.6 Additional Services

Depending on the available services from the cloud provider, additional functionality may need to be created as part of the deployment (e.g., NFS). This is reflected within the `top.sls` file.

6.6.1 NFS service

When we started with Azure, there was no NFS service available. We needed to build an NFS service with the tools we ship in SUSE Linux Enterprise Server for SAP Applications. As the NFS service should be highly available, we also needed a second virtual machine to build a two node cluster.



Over time, Azure has provided more services. At the time of this writing, there is a native NFS service (Azure Netapp files - ANF). The Azure file service is also being extended with similar functionality.

If the cloud native NFS service is used, no additional virtual machines will be created and the native service need to be set up in advance.

6.6.2 Fencing service

In high availability clusters, a so-called "split-brain" condition occurs when cluster nodes can no longer communicate with each other. This is a serious situation that can result in transaction inconsistencies as each node continues to write data. Thus, a mechanism, called 'fencing', is needed to switch off or reset one machine until synchronization can be restored.

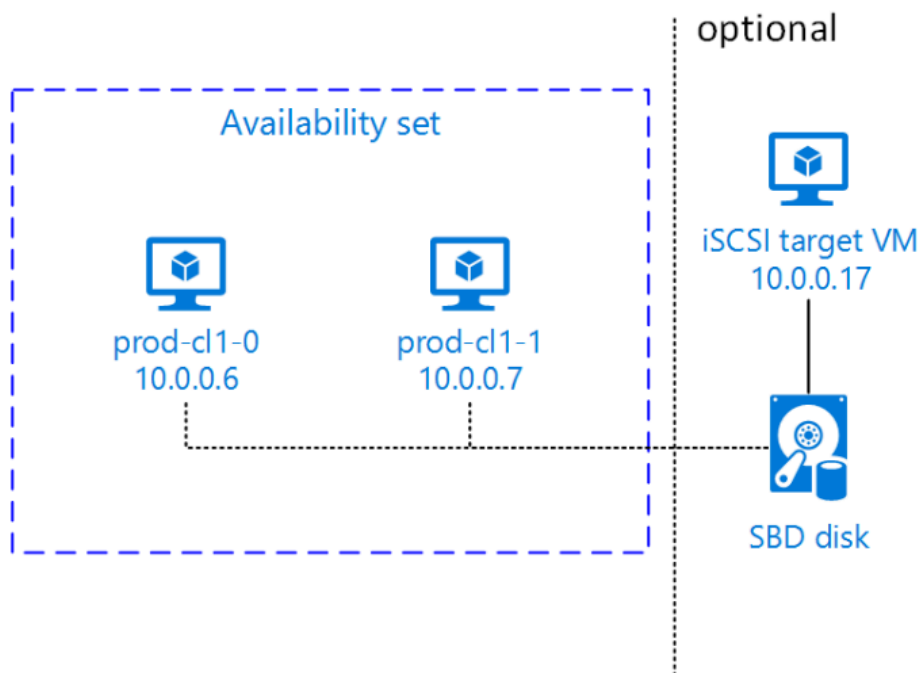
There are several methods which can be used, depending on the capability of the cloud provider. Microsoft supported SUSE clustering as the first Linux HA solution on Azure. Microsoft and SUSE created a fencing agent for the cluster for Azure. This fencing agent should remove a machine as quickly as possible (immediately) from the cluster to ensure that there is only one active node and avoid data corruption.

Initially, the Azure infrastructure only provided a way to gracefully shutdown a machine, which took 10 to 15 minutes. This is too long for the split-brain fencing requirement.

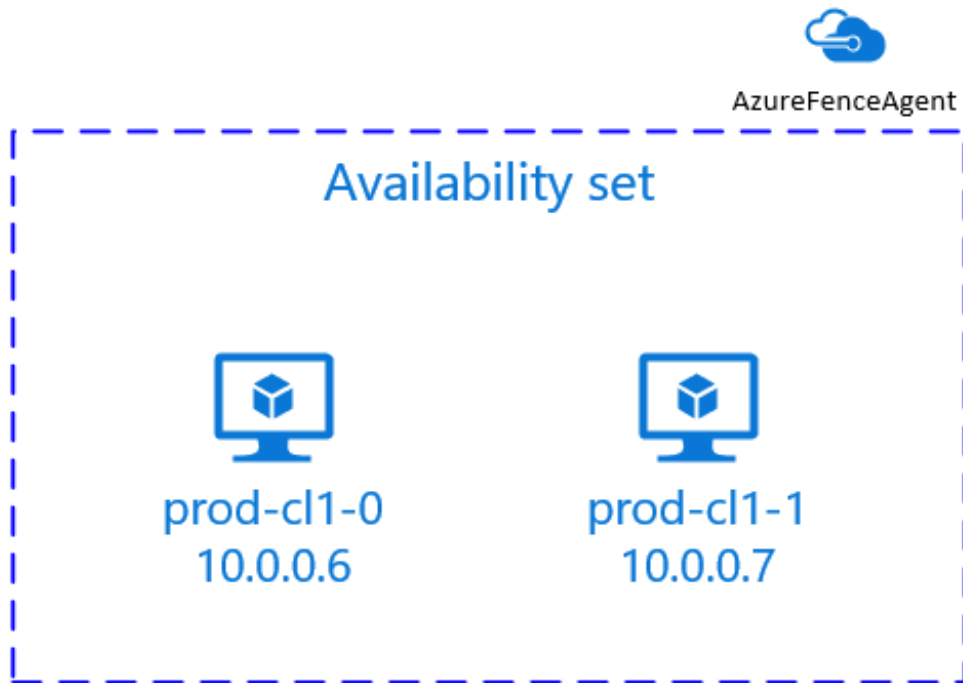
To improve on this time, SUSE implemented an OS native mechanism to fence virtual machines. This technology is provided within the SUSE HA Extension and uses storage as an additional communication layer between the nodes. This requires a raw shared disk, a central place where both nodes can write messages. This is called 'SBD' or Stonith Block Device or Split Brain Detector.

When originally desinging this solution, the Azure infrastructure did not provide a raw disk service that could be shared between nodes. Therefore, it needed to be built with the Linux tools available in the SUSE Linux distribution. With help of an iSCSI server and the Linux watchdog, SBD provides a fast and reliable fencing mechanism.

This requires one additional server to provide a iSCSI service.



Following recent improvements, there is now a method in the Azure API to "kill" a virtual machine. The fencing agent can make use of this, and no additional iSCSI machine is needed. However, the drawback of using the Azure API for this is that a public network connection is needed.



So, you can choose between two methods:

- SDB fencing with the help of an iSCSI service
- Agent based fencing through API access

In the meantime, there is a third option. Azure also provides a raw shared disks as a native service. As of the time of writing the document, only the SBD-fencing mechanism is implemented within the automation.

! Important

A working STONITH method is mandatory to run a supported SUSE cluster!

6.6.3 Monitoring service

If the Monitoring Service is to be deployed as part of the Automation, an additional virtual machine to support the Prometheus and Grafana services used to provide this capability.

7 Physical

The SAP automation consists of several building blocks, this section is in two parts, infrastructure deployment with Terraform and configuration management with Salt.

As the project is under active development to improve it and make it simpler to use, this document focuses on the project version 6.0.0 for the Terraform section and v6 of the rpm packages for Salt formulas. New versions may have additional features or different files than shown here, but the general guidelines should still be applicable. ---

7.1 Prerequisites

First make sure that all prerequisites are met:

1. Have an Azure account
2. Install the Azure command line tool *az*
3. Install *terraform* (v12) (it comes with SLES within the public cloud module)
4. Download the SAP HANA install media
5. Create 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 HANA install media (if required)

SUSE recommends to use the following directory structure:

```
version
├── SWPM
│   ├── SWPMxxxxx
│   └── SAPCAR_xxxx
│
├── EXPORT
│   ├── xxxxEXPORT_1.zip
│   └── xxxxEXPORT_2.zip
```

```

|   L...
|DBCLIENT
|   LIMDB_xxxx.SAR
|
|BASKET
|   |SAPHOSTAGENTxxxx.SAR
|   |igshelper_xxxxxx.sar
|   |igsexe_xxxxx.sar
|   |SAPEXEDB_xxx.SAR
|   |SAPEXE_xxx.SAR
|
|HANA
|   |xxxxxxxxx_part1.exe
|   |xxxxxxxxx_part2.rar
|   |xxxxxxxxx_part3.rar
|   |xxxxxxxxx_part4.rar

```

BASKET : contains SAP kernel, patch + more, like the hostagent.
 DBCLIENT : contains the package corresponding to DB CLIENT, e.g., HANA
 EXPORT : contains the package corresponding to EXPORT files
 SWPM : contain the corresponding files of SAPCAR and of the SWPM
 HANA : contain the full HANA media

7.2 Get the project

The project is hosted on a public GitHub site where it can be downloaded to your local machine.

<https://github.com/SUSE/ha-sap-terraform-deployments> ↗

The project has 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 brief but important documents for certain parts of the solution.

The directory *generic_modules* provides modules which are used by all cloud vendor templates. That includes common variables, locally 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 part of the Salt configuration management.

7.3 Terraform Building Blocks

Terraform relies on 'providers' to interact with remote cloud frameworks.

Providers are plugins and released independently from Terraform itself, this means 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.

Switching into a *cloud provider* directory, shows one directory *modules* and several *.tf* files which together build the Terraform template.

When creating Terraform configurations, best practice is 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 relevant provider module of Terraform and defines all the needed cloud specific entities.

The *main.tf* file

provides all the values for the 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 that represent the technical building blocks in the project.

The *output.tf* file

provides the values returned from the modules; i.e., to be used or displayed.

terraform.tfvars

is a variable definitions file which will get automatically consumed. This is used instead of providing values manually. It is the main configuration file and should be the only Terraform file which requires modification.

terraform.tfvars.example

is an example configuration file with many pre-filled values to set up the solution. **It can be used as a starting point for your own file.**

variables.tf

provides all 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 what is possible.

E.g., the variable *provisioner* is like a switch to run either the Salt or Terraform portion only

A *module*

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

Modules are used as part of the technical building blocks; e.g., a HANA node.

The module directory consists of *main.tf*, *variables.tf*, and *outputs.tf*.

These are the recommended filenames for a minimal module, even if they are empty.

main.tf is the primary entrypoint for defining the infrastructure building block.

There is one additional file, *salt_provisioner.tf*, which is responsible for handing over the needed values to the Salt building blocks. This is achieved by using a special Terraform resource called *null_provider*, which remotely runs the Salt pillar to configure the instances and execute the application installation for the building block.

7.4 Simple Install

SUSE provides example Terraform template and Salt pillar files to provide an easy way to perform an initial simple deployment.

1. Open a browser and goto <https://github.com/SUSE/ha-sap-terraform-deployments> 
2. Click on *tags*
3. Click on *6.0.0*

What is new and what has changed can be seen from this page. If older versions of the project are used, be sure to carefully review and understand the differences.

The *Usage* section provides you with a link to an OpenBuildServer (OBS) repository where the RPM packages of the building blocks discussed above are stored. Each project version has a unique repository.

The value/link to the repository will need to be included within the Terraform variables (terraform.tfvars) file. So copy the line as described.

4. Next go to *Assets* and download the *Source code* as *.zip* or *.tar.gz*
5. Extract it into a folder on the local computer
6. Go to this folder and into the subfolder for the cloud provider
7. Copy the file *terraform.tfvars.example* to *terraform.tfvars* There are many key-value variable pairs, some enabled and some disabled with a *=* or a *#* in front. In order to perform a simple deployment, only update the parameters as listed below.
8. Change the region in which to deploy the solution, change *az_region* = "*westeurope*" to the Azure region required.

1. To make it easier to start, change all 4 image 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., replace

```
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 all needed SUSE repositories attached automatically.

Next, set the name of the *admin_user* to the name you want to use.

2. The next step is to provide ssh keys to access the machines that will be deployed.

SUSE recommends creating new sshkeys for the deployment. Both public and private keys will need to be provided, as they are copied to the cluster nodes during the deployment. Change the two location variables and point them to your files.

3. As the SAP Install Media is needed for the automatic deployment of HANA, an Azure storage account needs to be created. The SAP HANA media will need to be copied to this storage location. If the SAP media is already extracted this will save time during the deployment.

Next, provide the name, key, and path to this storage account, change:

```
storage_account_name
storage_account_key
hana_inst_master
```

The inst_master variable should point to the directory where you have the extracted the hana install files. There are more possibilities, but, for simplicity, have everything already extracted on your share.

Disable the other hana variables by adding a '#' in front of them:

```
#hana_archive_file = "IMDB_SERVER.SAR"
#hana_sapcar_exe = "SAPCAR"
#hana_extract_dir = "/sapmedia/HDBSERVER"
```

4. Additional ssh keys are needed 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
```

5. Open the tfvars file again to make final changes.

To create a HANA Systemreplication HA automation, uncomment:

```
#hana_ha_enabled = true
```

by removing the #.

To create a cluster, we need to enable a few other services. Uncomment:

```
#hana_cluster_sbd_enabled = true
```

by removing the #.

6. Now we need to point to 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"
```

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

```
#monitoring_enabled = true
```

8. As the last step, we enable a simplification parameter which tries to determine 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.

First, go one directory up, change to the *pillar_example* directory, and then change to the *automatic* directory. Here you can see 3 additional directories. They will provide the configuration variables for the relevant services. This automatic folder will work for all cloud providers we support today.

It is possible to change other settings (e.g., passwords), but, for a simple test, do not modify these values.

Save any changes to the file and go back to the main directory.

We are ready to run Terraform.

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

If all goes well, after ~40 minutes (depending on the speed of the instances) you will have an installed and running HANA System Replication Cluster.

As a jump host with a public IP address is created as part of the deployment, it is possible to log in to any virtual machine as part of the deployment 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. This is computed in other terraform files and some *local* variables, used for mainly for the autogeneration of the network. In addition, it provides the *resources* for the network setup with the virtual network, the subnet and routing, the resourcegroup to be used, a storage account, all the network security groups (nsg), and definition of the jump host.

The *main.tf* file is the main file and calls child modules, which consist of the various building blocks and the required input and output variables defined by the child modules. In addition, it provides the calculation for the autogenerated IP addresses.

There is the (default) possibility to autogenerate network addresses for all nodes. For this, it is important to remove or comment out all the variables related to the IP addresses (more information in variables.tf). With this approach, all the addresses will be retrieved based on the provided virtual network address 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

Name	Terraform variable	IP Address	Comment
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
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

In order to reuse existing network resources (virtual network and subnets), configure the *terraform.tfvars* file and adjust the relevant variables.

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

! Important

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

The *output.tf* file is a way to expose some of the internal attributes. These 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 it provides all required Terraform input variables and their default values within the solution instead of having them in the main.tf file.

As there are many variable values to input, these need to be defined in a variable definition file named *terraform.tfvars*. Terraform will automatically load the variable values from the variable definition file if it is named *terraform.tfvars*.

The *modules* directory provides 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 *salt_provisioner.tf* file sets the **role** of the **node** and, with the help of a Terraform file provisioner, will pass the needed variables which were set in Terraform **as custom Salt grains for the node** and starts the Salt provisioning process.

7.4.2 SAP Sizing

One of the key points to consider in an SAP deployment is sizing and applies across three key areas: compute power, storage space and I/O capacity, and network bandwidth.

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 with 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 continuous process to translate your business requirements to the correct (virtual) hardware resources.

This is a mandatory step and should not be underestimated.

SUSE makes it easier to deploy the right instance sizes with the right disks types and performance, as well as the right network settings. A simplified SAP sizing has been introduced with 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

It is possible to customize the settings within the *terraform.tfvars* file, or provide a permanent solution in the variables file.

The Demo and Small size are designed for non-production scenarios and do not use SAP certified instance types, whereas the Medium and Large are meant for production usage and therefore use SAP certified instance types. The setups also use the correct 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 be kept in mind when deciding on a instance type.

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

The maps below describe 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 needs to be used.

disks_size

The size of the additional disk is expressed in GB. Every size has certain IOPS caps.

caching

The caching recommendations for Azure premium disks assume the I/O characteristics for SAP HANA, as follows:

- /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 implies, 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. At a minimum 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. The recommendations are:

- 256 KB for /hana/data
- 64 KB for /hana/log

Name of the VolumeGroup

This is the name of the volume group used.

Mount path

This is the mount point where the volume gets mounted.

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

(hash character)

is used to split the volume groups.

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

, (comma)

is used to define the logical volumes for each volume group.

names

The names of the volume groups (e.g., datalog#shared#usrsap#backup#sapmnt).

luns

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

sizes

The size dedicated for each logical volume and folder (e.g, 70,100#100#100#100#100).

paths

Folder where each volume group will be mounted (e.g., /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 a change to the default (demo) is needed or, better still, 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 required services for the SAP business applications and always needs a database.

It is the overall task of sizing to fulfil the requirements of Netweaver plus the database, and this is what is combined within the T-Shirt sizes of the solution.

Details of the solution T-Shirt sizes are provided below.

7.4.2.2.1 Demo

HANA instance size

Standard_E4s_v3

Accelerated networking

false

TABLE 7.2: 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 7.3: 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

true

TABLE 7.4: HANA DISK CONFIGURATION DETAILS

disks_type	Premium_LRS,Premium_LRS,Premium_LRS,Premium_LRS,Premium_L-RS,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 7.5: 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

Accelerated networking

true

TABLE 7.6: HANA DISK CONFIGURATION DETAILS

disks_type	Premium_LRS,Premium_LRS,Premium_LRS,Premium_LRS,Premium_LRS,Premium_LRS,Premium_LRS,Premium_LRS,Premium_LRS,Premium_LRS
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 7.7: 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

Accelerated networking

true

TABLE 7.8: HANA DISK CONFIGURATION DETAILS

disks_type	Premium_LRS,Premium_LRS,Premium_LRS,Premium_LRS,Premium_LRS,Premium_LRS,Premium_LRS,Premium_LRS,Premium_LRS
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 7.9: 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

Resources are the most important elements in Terraform. There is another resource type used as last a 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.

Finally, 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*.

From this point on, all work is performed on the respective node itself.

7.5.1 Our Architecture for the Salt building blocks

shaptools

low level python wrapper (API) around SAP utilities and commands

Execution module

provides the methods in the lower layer (shaptools) to Salt

State

combination of execution modules and other parts with logic to define a specific configuration

Formula

group of states that give a context for building blocks; e.g., HANA

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

1. Bootstrap Salt installation and configuration.
2. Perform OS setup operations; register to SCC, if needed; update the packages; etc. by executing the states within `/srv/salt/os_setup`.
3. Perform predeployment operations by execution of the `/srv/salt/top.sls` states. It updates hosts and hostnames, installs the formula packages, etc.
4. Perform deployment operations depending on the overall configuration settings; e.g., install SAP applications, configure and setup HA with the salt formulas.

7.5.2 Salt Overview

The SAP building blocks are created with help of Salt formulas after provisioning the virtual machines with Terraform. The formulas are shipped as RPM packages with SUSE Linux Enterprise Server for SAP Applications.

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

In this automation solution, we use the Salt minion option. The steps in the formulas must be executed in all of the minions and are performed through a SSH connection.

The core of the Salt State system is the SLS, or SaLt State file. The SLS is a representation of the state in which a system is expected to be, 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, called a state tree, that contains state files.

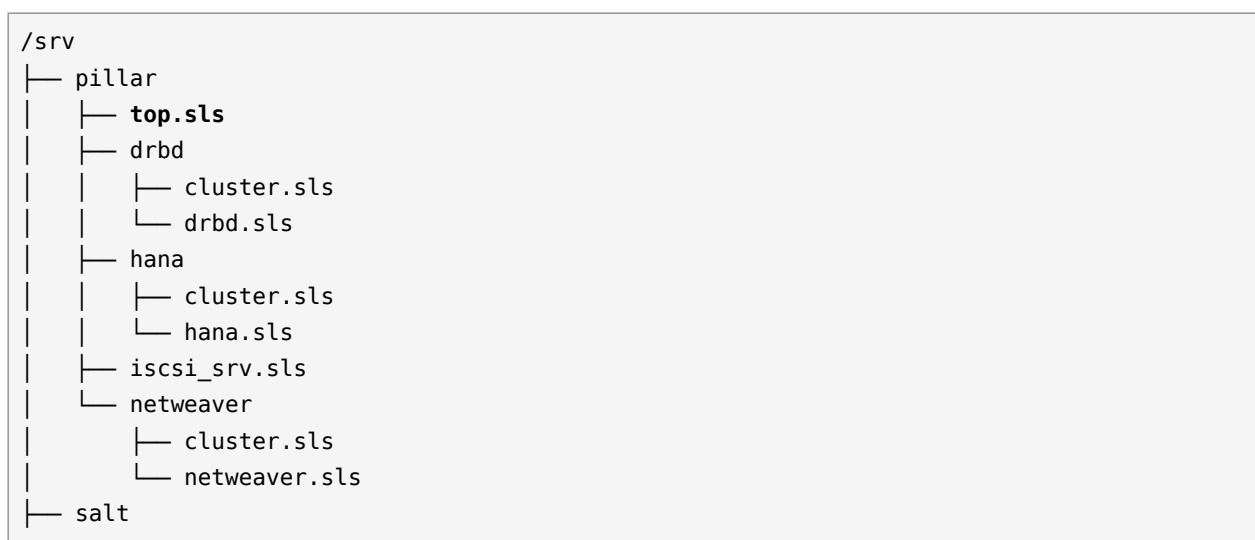
7.5.2.1 Salt pillar

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



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

State top.sls file

```
base:
  'role:iscsi_srv':
    - match: grain
    - iscsi_srv

  'role:hana_node':
    - match: grain
    - hana.hana

  '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, 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, provide your own pillar data files. There are some basic examples within the directory *MAIN/pillar_example*. As the pillar files provide data for the Salt formulas, all of the possible pillar options can be found in each formula project.

Important

Pillar files are expected to contain private data, such as passwords, required for automated 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 configuration of GPG keys and procedure for pillar encryption are described in the Saltstack documentation guide:

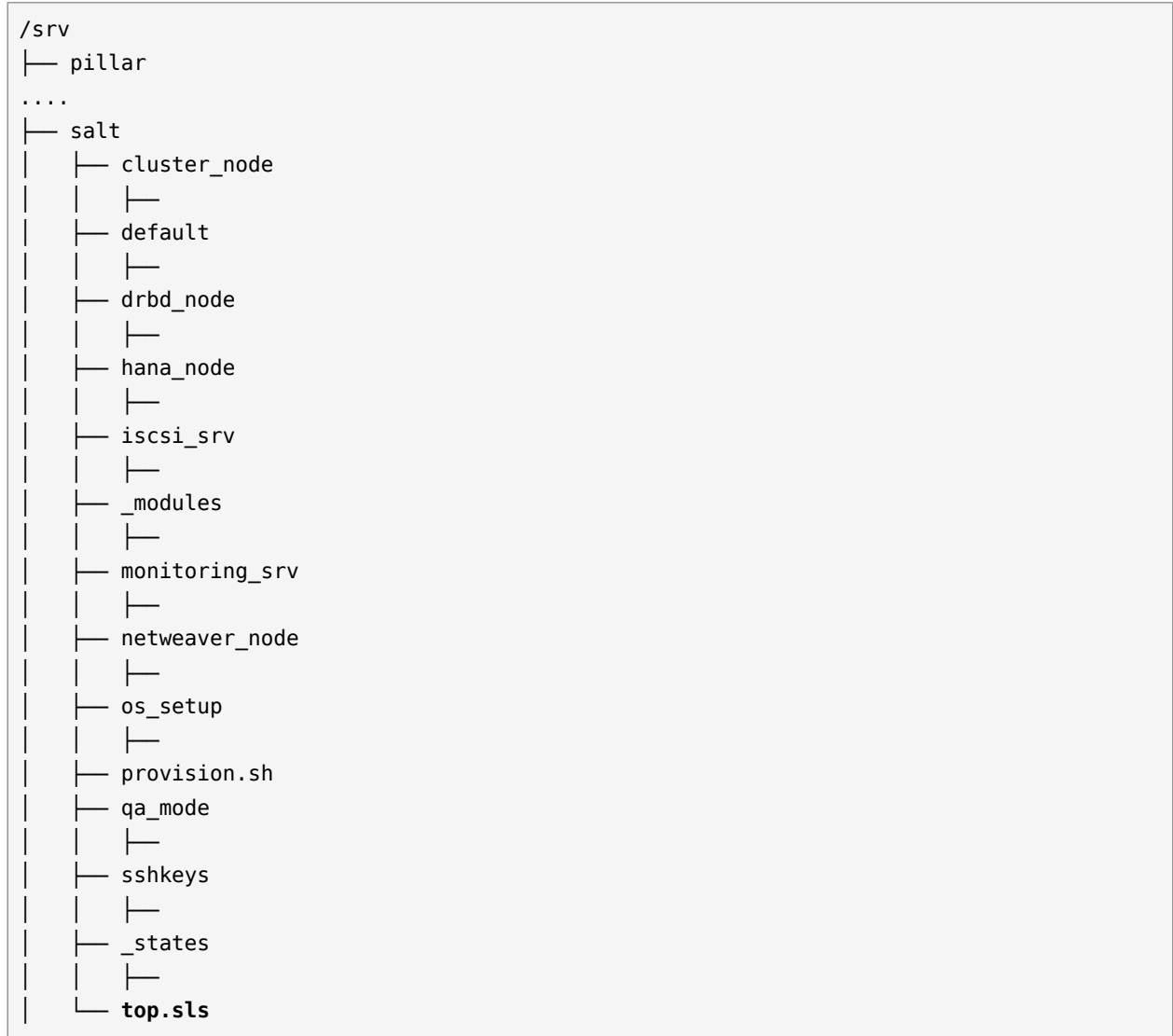
1. SaltStack pillar encryption (<https://docs.saltstack.com/en/latest/topics/pillar/#pillar-encryption>) 
2. SaltStack GPG RENDERERS (<https://docs.saltstack.com/en/latest/ref/renderers/all/salt.renderers.gpg.html>) 

Encryption is not included in this automation solution. You are strongly advised to take appropriate security precautions.

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



Within this directory structure, all needed steps that depend on the *role* of the node can be seen. The *top.sls* file describes two environments for the nodes, *pre-deployment* and *base* which reflect steps 3 and 4 of the workflow above.

The Pre-deployment environment is needed, as formulas can not be installed and used 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

base:
  'role:hana_node':
    - match: grain
    - hana

  'G@role:hana_node and G@ha_enabled:true':
    - match: compound
    - cluster

  '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, such as the operating system, domain name, IP address, kernel, OS type, memory, and many other system properties. The SUSE Automation project uses custom grains to match the roles and the further states.

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



Caution

If using the Salt formulas independently from the Terraform templates, it is important to take care of providing all required variables that would normally get set by the *salt_provisioner.tf*.

7.5.2.4 State details

If targeting 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* directory

```
| | | os_setup
| | |   | init.sls
| | |   | ip_workaround.sls
| | |   | minion_configuration.sls
| | |   | packages.sls
| | |   | registration.sls
| | |   | repos.sls
```

there is one interesting file, the *minion_configuration.sls*. It provides the configuration how and where Salt / the Minion looks for Salt states and Salt formulas.

Looking deeper into one of the directories, *hana-node*, there are more files.

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

```

When targeting 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. Salt executes what is in *init.sls* in the order listed in the file. When a 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 gets the SAP HANA Media with help of *hana_inst_media*, creates the mountpoints, partitions disks for SAP HANA, and enters 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*. It then installs 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 they 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 the sane defaults and no additional configuration.

The formulas in the automation solution are configurable by including data in *Pillar* files, as discussed above. During RPM install, the files of the packages end up in the directory */usr/share/salt-formulas/states*. This was defined as the directory where Salt searches for files in addition to */srv/salt* (see *os_setup* state above).

shaptools package. The directories *modules* and *states* come from the install of the package **shaptools** and provide a python wrapper as an API for sap command line tools, making it simpler to with Salt. This package is a base dependency for most of the SUSE formula packages as it provides the needed SAP commands.

```
| | | _modules
| | |   |
| | |   | _...
| | |   |
| | | _states
| | |   |
| | |   | _...
```

7.5.3.1 HANA formula

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

The structure is similar to what has been 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 to 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. To install it manually please use zypper, as this will include the other dependent packages such as salt-shaptools and habootstrap-formula

```
zypper install saphanabootstrap-formula
```

The Salt formula will need input data through a pillar file which is part of the main project file (in MAIN/pillar/... or on the node /srv/pillar) If using the formula standalone, the data needs to be provided manually. There are more options available as shown in the example file.

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:
          key_name: 'backupkey'
          environment: 'hana01:30013'
          user_name: 'SYSTEM'
          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 formula is executed within the *hana_node* Salt state files.
2. If wanting to execute the formula manually

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

With the help of the pillar data, the state file, and the formula, Salt will create all needed configuration on the node, will install HANA and, if enabled, will install hana systemreplication and set up the pacemaker cluster, correctly 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 practices guides SUSE created with the Cloud provider 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 takes care of the ASCS, the Application Servers, and, if HA is selected, the Enqueue Replication server.

The formula has some **hard dependencies** and **all of them must be in place** for a successful netweaver deployment. In order to deploy a valid Netweaver environment, a NFS share is needed (SAP stores 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 (e.g., /sapdata/HA1/sapmnt and /sapdata/HA1/usrsapsys). This content is removed by default during the deployment.

Secondly, the SAP installation software (SWPM) must be available in the system. To install the whole Netweaver environment with all the 4 components, the SAP Media must be provided. The structure depends on the version of SWPM.

For SWPM 1.0 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, such as 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 has been illustrated 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 described earlier, a pillar file is needed with the configuration. There is one example in the path, which can be used as a base for standalone Salt usage. In general, the pillar data will be passed from the Terraform main project.

As SAP Netweaver has additional nodes in an HA environment, the pillar file will be larger than the one for HANA. Take the time to review this by viewing the example file.

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

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

In addition, here are the templates which are used by SWPM for an automated hands-free installation of the SAP Netweaver services.

7.5.4 High Availability formula

The *habootstrap-formula* will take care of the needed cluster setup for SAP HANA, SAP Netweaver, and, if needed, for the HA NFS service built with DRBD.

The formula will be similar to all the other formulas used and 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.sls
```

The main difference to the HANA and Netweaver formula is that the *init.sls* already makes use of *jinja*. Jinja is the default templating language in SLS files and gets 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 additional services depend on what is used or available from the cloud provider, 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 described *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 atform. It is implemented as a kernel driver, several userspace management applications, and some shell scripts. So think about it as "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 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 between 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 to 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 fencing method depends on the cloud provider's features. 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 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

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

Copyright © 2006–2021 SUSE LLC and contributors. All rights reserved.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or (at your option) version 1.3; with the Invariant Section being this copyright notice and license. A copy of the license version 1.2 is included in the section entitled "GNU Free Documentation License".

SUSE, the SUSE logo and YaST are registered trademarks of SUSE LLC in the United States and other countries. For SUSE trademarks, see <https://www.suse.com/company/legal/> .

Linux is a registered trademark of Linus Torvalds. All other names or trademarks mentioned in this document may be trademarks or registered trademarks of their respective owners.

This article is part of a series of documents called "SUSE Best Practices". The individual documents in the series were contributed voluntarily by SUSE's employees and by third parties. The articles are intended only to be one example of how a particular action could be taken.

Also, SUSE cannot verify either that the actions described in the articles do what they claim to do or that they don't have unintended consequences.

All information found in this article has been compiled with utmost attention to detail. However, this does not guarantee complete accuracy. Therefore, we need to specifically state that neither SUSE LLC, its affiliates, the authors, nor the translators may be held liable for possible errors or the consequences thereof. Below we draw your attention to the license under which the articles are published.