SUSE SAP automation solution

SUSE SAP automation solution

Abdelrahman Mohamed

Publication Date: 2021-01-19

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

*

Contents

I SUSE SAP AUTOMATION SOLUTION 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

4.3 Roles and Collaboration 9

4.4 Processes and Functions 10

4.5 Factors, Flavors and Deployment Types 10

Factors 10 • Flavors 10 • Deployment Types 10

5 Application 11

5.1 SUSE Linux Enterprise Server for SAP Applications 11

5.2 SAP Application 11

5.3 Presenting the SAP Media 11

5.4 Terraform 12

5.5 SALT 14

Netweaver 15 • HANA 15 • HA 16

5.6 Monitoring 17

SAP HANA Database Exporter 17 • High Availability Cluster Exporter 18 • SAP Host Exporter 18

6 Technology 19

6.1 Terraform 19

6.2 Salt 19

SAP Sizing 20 • Building Blocks 20 • High Availability 20 • Additional Services 21

7 Physical 23

7.1 First make sure that all **pre-requirements** are met: 23

7.2 Get the project from the GitHub site 24

7.3 Terraform Building Blocks 24

7.4 Simple Install 26

Terraform file details 28 • SAP Sizing 31

7.5 Salt Building Blocks 31

Our Architecture for the Salt building blocks 32 • Salt Overview 32

I SUSE SAP automation solution

1 Preface 2

2 Introduction 3

3 Strategy 5

4 Business 7

5 Application 11

6 Technology 19

7 Physical 23

*

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
- migration

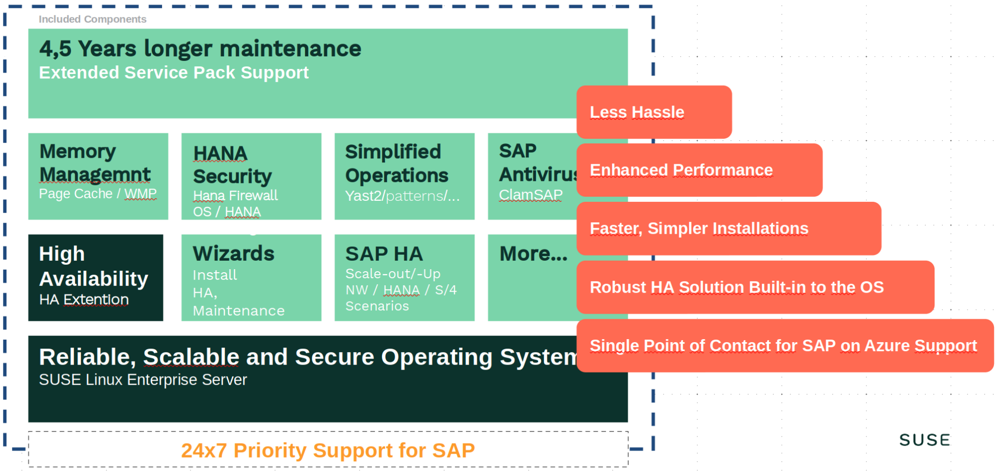
1 link: Archimate Enterprise Architecture (https://pubs.opengroup.org/architecture/archimate3-doc)

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 public 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 multiple environments, for multiple 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 the environment after the install is complete.

To help with this, SUSE has added the ability to 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 has 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 and 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 SAP services are deployed on-premises with well-established procedures, but it is important to plan and size the required hardware several years ahead and estimate the maximum workload e.g. for the next 5 years.

It is difficult to predict the future, so when considering requirements such as capacity, as there are many factors which may affect this over the system lifespan, the values selected for this often ends up being a 'best guess'.

Today, with quickly changing environments, many businesses need to accelerate innovation and increase agility across their business. This allows them to achieve a faster time-to-market in addition to lowering costs.

One key example here is that you no longer need to plan the hardware sizing for the next 5 years, as a larger, faster, or even smaller infrastructure is only one reboot away.

"Rightsizing" or infrastructure optimization remains an important task and should be front of mind when managing an SAP Landscape in the cloud. Not taking care of this cloud benefit could have large cost implications, companies that “rightsize” their workloads are often able to cut costs by of 30-60%.

3.1 Context

There are many benefits gained when moving your 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 set-up takes days and sometimes months to order it and set up. With the cloud, IT teams can easily configure and manage the setup remotely.
This is the area where SUSE’s automation solution for SAP can help.

Reduce Costs

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). Costs can be controlled as the cloud model is a ‘pay as you use’ scenario and many cloud vendors provide several consumption options to provide more choice to their customers.

This is an area where SUSE can help together with the cloud providers to offer the right options, but this also means you need to adapt to how you use the resources and the SAP software.

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 as mentioned above, "rightsizing" and instance optimization to provide an efficient solution.

Maintenance

With the public cloud, IT departments no longer have to worry about managing and maintaining the hardware and underlying infrastructure, the cloud service provider handles this for your organization and frees up company resources to focus on other business activities.

Resiliency

For businesses these days, 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. Cloud providers offer a variety of services to help protect businesses from any security threat or outage.
SUSE's SAP HA Automation can add to these services to provide less downtime of your SAP application.

Remote access

With the cloud, your employees can access data from anywhere, and at any time via the internet making business more flexible and increase the productivity of the employees.
SUSE products natively provide many options for remote access and control.

So overall, public clouds offer significant benefits for all customers regardless of size. The use of public cloud resources can lower infrastructure costs and improve the scalability, agility, flexibility and availability of 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 specialised knowledge is no longer needed. Functional teams still exist and will need to work together, this is often best achieved with a DevOps approach utilising Infrastructure-as-Code (IaC).

This means that when implementing SAP in the cloud, knowledge will be required of the cloud infrastructure and the various possibilities that 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 needs to be fine-tuned based on customer requirements around system availability i.e 99.99%, 99.95% or 99.9%. Single Point of Failures (SPOF) in the components and services will need to be idenified and protected against, this is normally achieved with an HA Cluster. Other SPOFs within the infrastructure will need to be protected using 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, points 1, 2 and 4 are potential SPOF.

SUSE's SAP Automation will 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 (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 availablity.

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 continue 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. An SAP license for the SAP software to be deployed is required.
2. A understanding of the resource requirements of the SAP workloads (via an SAP System Sizing exercise).
3. Certified instance types must be used based on the capacity required by SAP software.
4. Ensure suitable network connectivity is provided (bandwidth, latency, and package loss) within the cloud environment for your SAP workloads.
5. Only deploy certified operating systems on which the SAP workloads will run.
6. A good operational knowledge of the Linux OS, SAP systems operations and the cloud infrastructure is needed.
7. Where highly available solutions are deployed, a deep understanding of the HA concepts and tooling along with how this functions within along side the resiliency capability of the cloud infrastructure is required.

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

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 it's 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

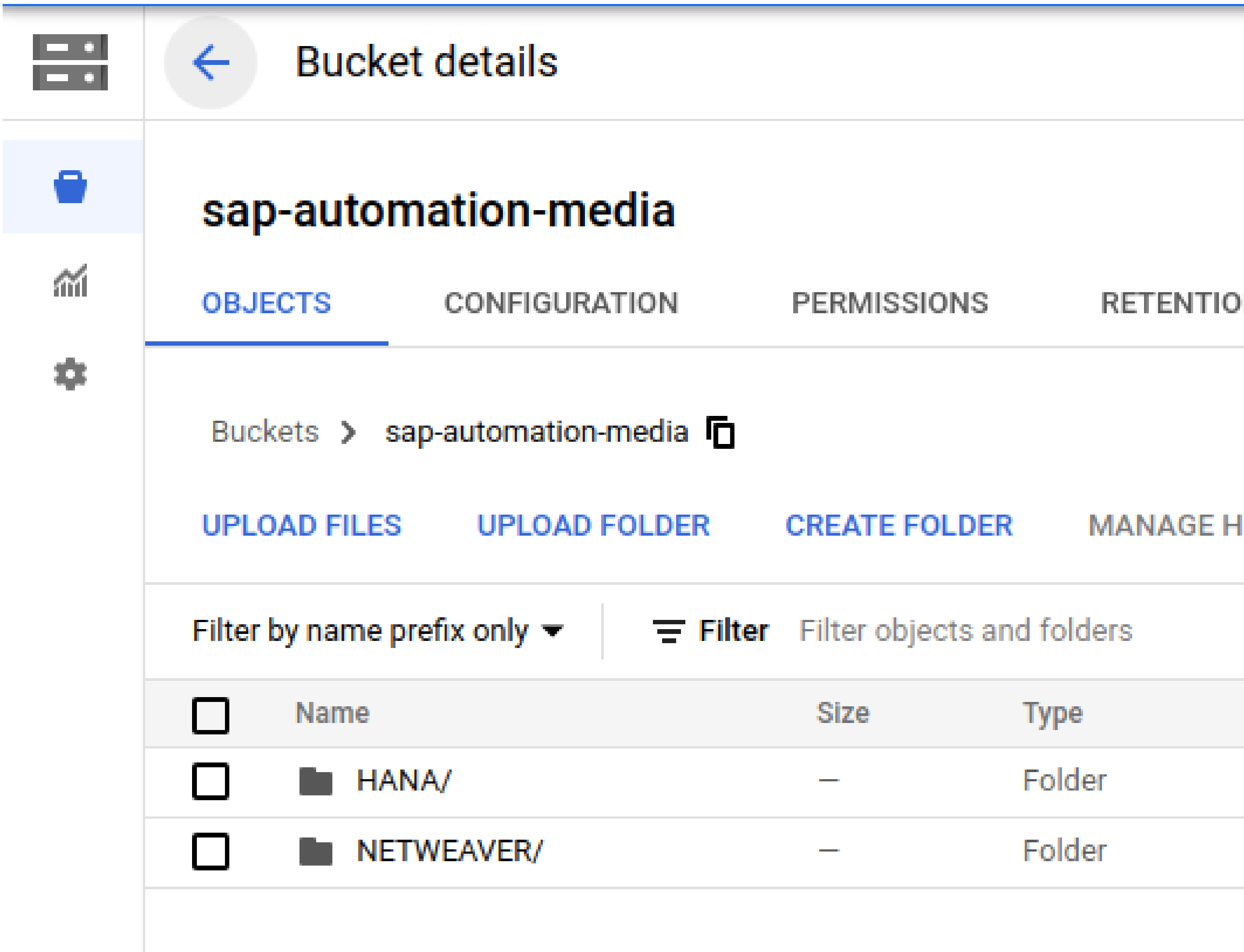
In order to use the automation project, there are preliminary steps which need to be taken. One of these is to prepare the SAP installation software. The SAP software can be downloaded from https://launchpad.support.sap.com/#/softwarecenter ↗, this will need to be performed manually before you start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

When deploying on GCP, a Storage Bucket is required to store the SAP Media. Using the GCP Console:

- Create a storage bucket.
- Create two folders (for the SAP HANA and SAP NetWeaver media) within the bucket.
- Upload the SAP media to the folder within the storage bucket.



TIP

The example shows a bucket called *sap-automation-media*, but a unique name should be used.

NOTE

For more information about how to create a Google Cloud Storage Bucket, refer to https://cloud.google.com/storage/docs/creating-buckets

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 imporant to note that Terraform can not simply create a landscape with a press of a button in another cloud. 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, whilst the code will need 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 which 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, GCP provides an easy to access web based command line shell (https://shell.cloud.google.com/) where Terraform is already pre-installed.

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

The suggestions from the Terraform files for the storage configurations are meant as good directions to start with. But you still should analyze 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, GCP 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 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 Client. All steps are executed on the Salt Master machine which sends instructions to the client to perform the required configuration actions.

Salt Client (Minion) only. All steps in must be executed on each individual Salt client where the formulas need to be executed. This is the approach used by the SUSE Automation framework as it removes the need for a central master system.

5.5.1 Netweaver

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

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

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

- Hostnames
- Virtual addresses
- NFS mounts
- Shared disks
- SWAP partition space

The Salt 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 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

5.5.3 HA

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

- Create and configure the High Availability cluster, pacemaker, corosync, GCE fencing and SAP resource agents.
- Adjustments for the Google CloudInfrastructure
- 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 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*

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

*

5.5.3.1 Other dependent 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

SUSE continually try 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 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 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

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 make creating the correct virtual machines, disks, networks etc. simpler.

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. Those 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.

6.2.1 SAP Sizing

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

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

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

As sizing is critical, it is important 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

The 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 within the Automation project is SAP HANA.

There are two possible models how SAP Business Suite can be deployed, 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 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.

The automation project uses the building blocks described before, to create the distributed landscape. It is described within the top.sls file, depending on the role of the node.

6.2.3 High Availability

There are two main building blocks which should be made highly available in order to achieve less downtime and eliminate single point of failures within the SAP Landscape. As there are many scenarios available from SAP, the Automation project supports the most popular scenarios.

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

Therefore, for each of these building blocks, one additional machine is required to build a two-node cluster within the HA Scenarios.

Within a Google Cloud SAP HA Cluster, the HANA Primary and Secondary nodes each reside in 2 different Availability Zones (AZs), therefore to provide an IP address which is portable between the 2 AZs, there are two available options:

1. A *Standard GCE Load Balancer* service from Google Cloud to provide traffic to only the active node.
2. An *GCE Overlay IP address*. It uses the *gcp-vpc-move-route* resource agent which can send network traffic to an instance, no matter which Availability Zones (and subnet) the instance is located in.

6.2.4 Additional Services

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

6.2.4.1 NFS service

Currently, we need 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 need two virtual machines to build a two-node cluster.

Note

Google Cloud provides a native NFS service (Google Cloud Filestore). It is planned to add the support for the Google Cloud Filestore service in the upcoming releases of the SUSE SAP automation platform.

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.

6.2.4.2 Fencing service

For high availability clusters, a mechanism to switch off or reset one machine in the case of a so called "split-brain" is needed. This is used when the 2 nodes can no longer communicate with each other. This action is called 'fencing'.

There are several methods which can be used depending on the capability of the cloud provider.

Google Cloud Supports the use of the GCE STONITH mechanism. This is shipped and supported with the SUSE HA Extension and has been specifically written to fence (poweroff/reboot etc) GCE instances as part of cluster operations.

The GCE fencing mechanism uses the cutting-edge Google Python APIs. There is no longer a need to install the Google Cloud SDK in each HA cluster node to enable the fencing functions.

> **Important**
>
> A working STONITH method is mandatory to run a supported SUSE cluster on Google Cloud.

6.2.4.3 Monitoring service

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

7 Physical

The SAP automation consists of several building blocks. First we want to look at the infrastructure deployment with Terraform and then to the Salt part.

Note

As the project is under active development to make it better and simpler to use, this document focuses on the project version 6.0.3 of the Terraform part and v6 of the rpm packages for Salt formulas. New version could have more features or slightly changed files as shown here, but the general guidelines should still be applicable.

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

1. Have a Google Cloud Platform account
2. Have a Service Account Key saved in the machine that will be used to initiate the environment.
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 a Google Cloud Storage Bucket
6. Copy or write down the name of the Google Cloud Storage Bucket
7. Copy the SAP HANA install media to the created Google Cloud Storage Bucket
8. Extract the HANA install media to the Google Cloud Storage Bucket

SUSE recommends to use 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 brief but important documents for certain parts of the solution

The directory *generic_modules* provides modules which are used by all cloud vendor templates, e.g. 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 'providers' to interact with remote cloud frameworks.

Providers are plugins and released independently 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, 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 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 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 gets 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. **You can use this as 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 whats 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 it's architecture, rather than directly in terms of physical objects.
Modules are used as part of the 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 is responsible for handing over the needed values to the salt building blocks. This is acheived by using a special terraform resource called *null_provider*, it remote executes the salt run to configure the instances and execute the application installation for the building block.

7.4 Simple Install

SUSE provide an 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*
   What's new and what has changed can be seen from this screen, so if older versions of the project are used, ensure 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 poject version has a unqiue repository.
   The value/link to the repository will need to be included within the terraform variables (teraform.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 your computer

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

7. Copy the file *terraform.tfvars.example* to *terraform.tfvars.example* There are many key-value variable pairs, some enabled some disabled with a = in front. To have a simple start, only modify what is described below

GCP

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

2. The next step is to provide ssh keys to access the machines which will be deployed. SUSE recommend to create new sshkeys for the deployent as both keys will need to be provided as they are copied to the cluster nodes. Change the two locations variables and point them to your files.

3. 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.

GCP

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. For a simple deployment, which uses only HANA, please switch to the *hana* directory and open the file *hana_sls*.

3. Change the PRIMARY_SITE_NAME to the desired value, along with value for the SECONDARY_SITE_NAME. 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 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 after ~40 Minutes (depends on the speed of the instances) you will have a installed and running HANA System Replication Cluster

GCP

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.

GCP

The *main.tf* file is the main file and calls the child modules which consist of the various 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 i is important 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).

GCP

Example based on `10.0.0.0/24` VPC address range. The virtual addresses must be outside of the VPC address range.

Name	Substituted variable	Addresses	Comments	
:---:	:---:	:----:	:---:	
Iscsi server	`iscsi_srv_ip`	`10.0.0.4`		
Monitoring	`monitoring_srv_ip`	`10.0.0.5`		
Hana ips	`hana_ips`	`10.0.0.10`, `10.0.0.11`		

Name	Substituted variable	Addresses	Comments	
Hana cluster vip	hana_cluster_vip	10.0.2.12	Only used if HA is enabled in HANA	
Hana cluster vip secondary	hana_cluster_vip_secondary	10.0.1.13	Only used if the Active/Active setup is used	
DRBD ips	drbd_ips	10.0.0.20, 10.0.0.21		
DRBD cluster vip	drbd_cluster_vip	10.0.1.22		
Netweaver ips	netweaver_ips	10.0.0.30, 10.0.0.31, 10.0.0.32, 10.0.0.33	Addresses for the ASCS, ERS, PAS and AAS. The sequence will continue if there are more AAS machines	
Netweaver virtual ips	netweaver_virtual_ips	10.0.1.34, 10.0.1.35, 10.0.1.36, 10.0.1.37	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) configuring the *terraform.tfvars* file and adjust the relevant 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 the 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.

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.

GCP

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

Formulas: 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 updates hosts and hostnames, installs the formula packages, etc
4. Execute deployment operations depending on the overall configuration settings e.g. install SAP applications and 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.

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

  '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, 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 provide data for the salt-formulas, you can find all of the pillar possible options in each formula project.

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)

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
│   │   ├──
```

```
|   |   ├─
```

```
|   ├─ sshkeys
|   |   ├─
|   ├─ _states
|   |   ├─
|   └─ top.sls
```

You will see within this directory structure all needed steps depending on the *role* of the node.

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
```

```
    - 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, 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 formulas 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

```
|   ├── 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.

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
```

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 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.

= = = = 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* files, what we discussed above. During RPM install, 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
|   |   ├── ...
```

= = = = = HANA formula

The main work of preparing the node for HANA and installing HANA is done by the *saphanabootstrap-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. 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 (MAIN/pillar/… or on the node /srv/pillar) If you use the formula standalone the data need to be provided manually.

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 formula is executed within the salt of *hana_node* state files
2. If you want to execute the formula manually

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

So with help of the pillar data and the state file and the formula, 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 GCP.

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 GCP for the HA scenario.

= = = = = 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 of a Enqueue Replication server.

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 than the one for HANA. 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 GCP 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.

= = = = 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.

= = = = Additional Services

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

= = = = = 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

= = = = = 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.

= = 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 7.1: SOLUTION ARCHITECTURE - FIXME MIGRATION

= = 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.

= = Appendix

*