Cluster Deployment Automation using Salt

Xabier Arbulu
Software Engineer
xarbulu@suse.com

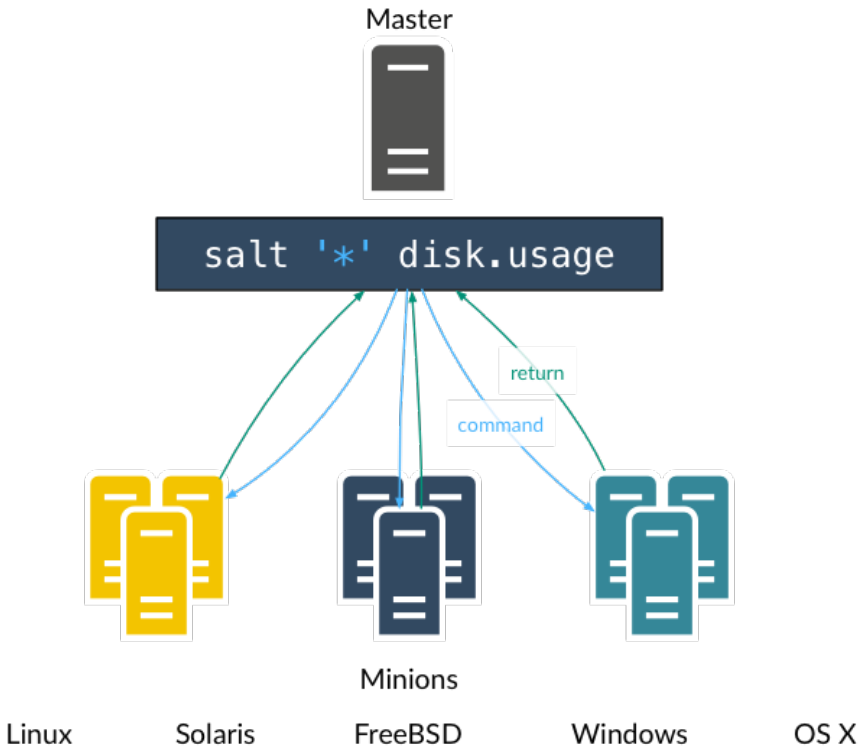
First of all, what the heck is salt?

Salt is...

- **A configuration management system, capable of maintaining remote nodes in defined states**
- **A distributed remote execution system used to execute commands and query data on remote nodes**
- **Has an extensive list of existing standard states and formulas for the many different purposes**

Source: https://docs.saltstack.com/en/latest/topics/#introduction-to-salt

To make it more visual...

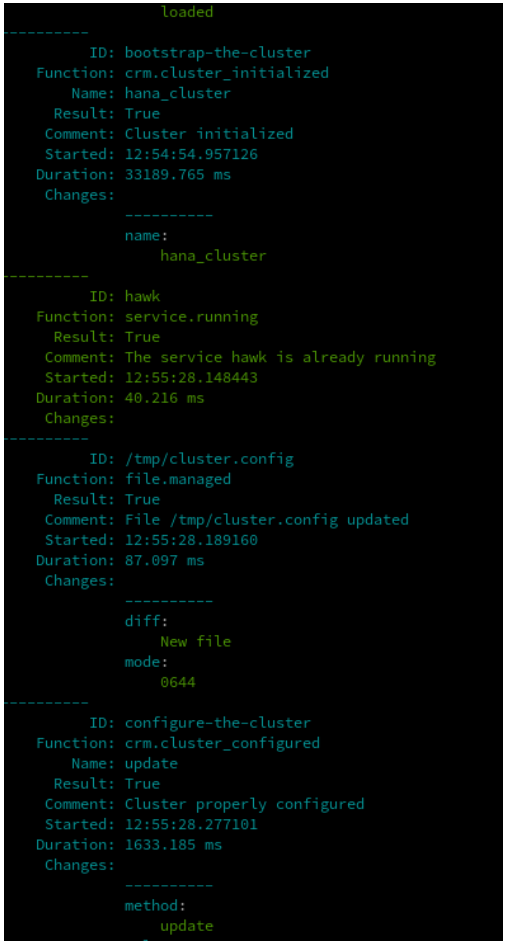


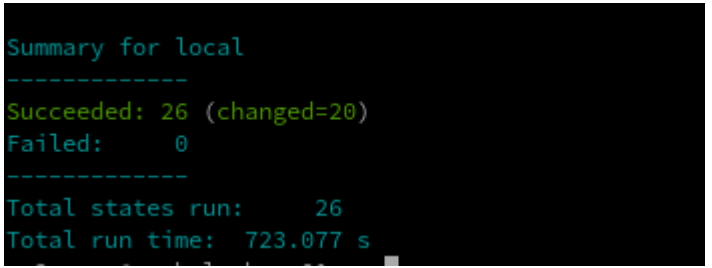
Salt is responsible for...

- **Send operations to the minions (in a master/minion architecture)**
- **Get the operations result and publish them**

Source: https://docs.saltstack.com/en/getstarted/images/remote-exe.png

And this is how it looks like...





And the magic word is…

Idempotence!

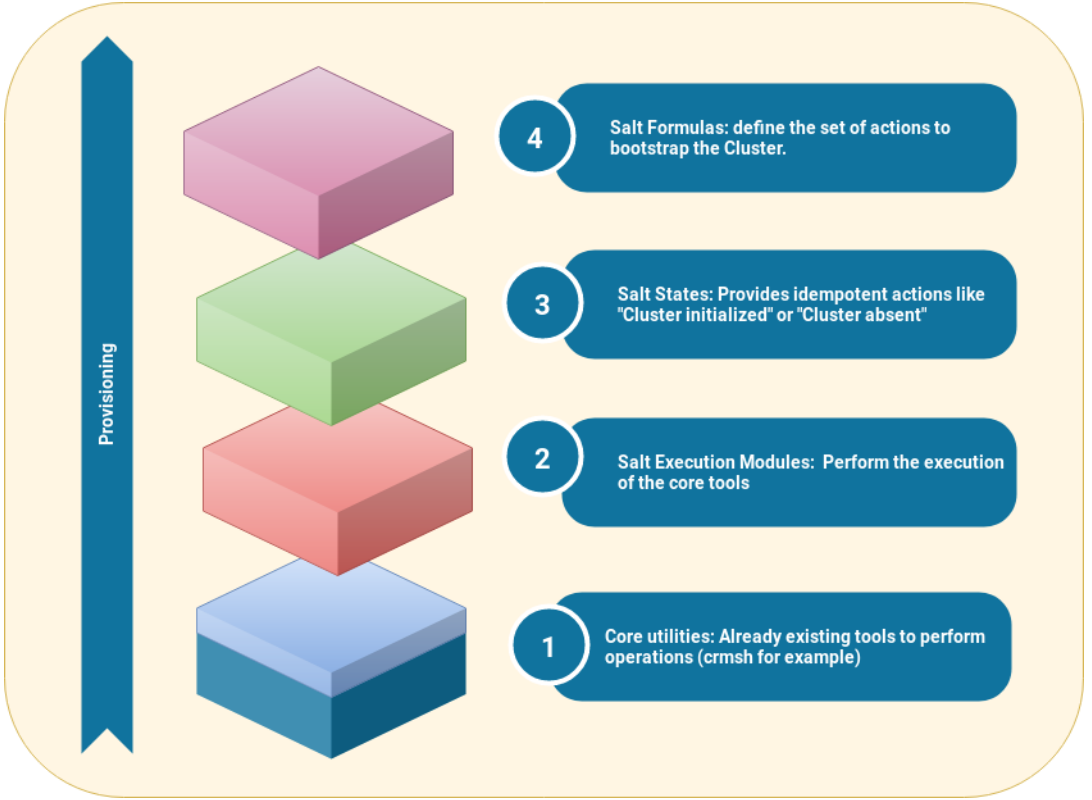
Idempotence (UK: /ˌɪdɛmˈpoʊtəns/,[1] US: /ˌaɪdəm-/)[2] is the property of certain operations in mathematics and computer science whereby they can be applied multiple times without changing the result beyond the initial application.

Basically, we want to have the same result always!

Source: https://en.wikipedia.org/wiki/Idempotence

Going deeper in our Salt projects

Architecture



Involved project

- **crmsh (as the core utility, pcs is not covered by now)**
  https://github.com/ClusterLabs/crmsh

- **salt-shaptools (salt modules and states python code*)**
  https://github.com/SUSE/salt-shaptools

- **habootstrap-formula (salt formula)**
  https://github.com/SUSE/habootstrap-formula

- **Drbd-formula**
  https://github.com/SUSE/drbd-formula

Python2 and Python3 are supported by now due some legacy code

Execution modules

Execution modules wrap simple (most of the times…) operations into the salt framework. They are straightforward actions and most of the times they return just a True/False or return code of the operations.

JUST DO IT!

```
ec2-user@xarbulu-hana01:~> sudo salt-call --local crm.status
local:
    0
ec2-user@xarbulu-hana01:~> sudo salt-call --local crm.cluster_status
local:
    0
ec2-user@xarbulu-hana01:~> sudo salt-call --local crm.configure_load update /tmp/cluster.config
local:
    0
ec2-user@xarbulu-hana01:~> sudo salt-call --local crm.cluster_init hana_cluster /dev/watchdog eth0
```

Execution modules

The current salt execution module wraps crmsh tools calls to perform almost all the operations (there are some few actions around corosync and its configuration files).

Most relevant methods:

- **status (*crm status*, check if the cluster is up and running or not in this node)**
- **start (*crm cluster start*, start the cluster in this node)**
- **stop (*crm cluster stop*, stop the cluster in this node)**
- **cluster_init (*crm cluster init*, initialize the cluster in this node)**
- **cluster_join (*crm cluster join*, join to an existing cluster)**
- **And much more, the implementation is a boring copy/paste most of the times**

States

Salt states are more complex scenarios where the final goal is to reach a specific/idempotent state about a particular topic. Most of the times they are composed with several execution modules combinations.

```
ec2-user@xarbulu-hana01:~> sudo salt-call --local state.single crm.cluster_initialized name=hana_cluster watch
local:
----------
          ID: hana_cluster
    Function: crm.cluster_initialized
      Result: True
     Comment: Cluster is already initialized
     Started: 13:09:16.625594
    Duration: 218.211 ms
     Changes:

Summary for local
------------
Succeeded: 1
Failed:    0
------------
Total states run:     1
Total run time: 218.211 ms
```

States

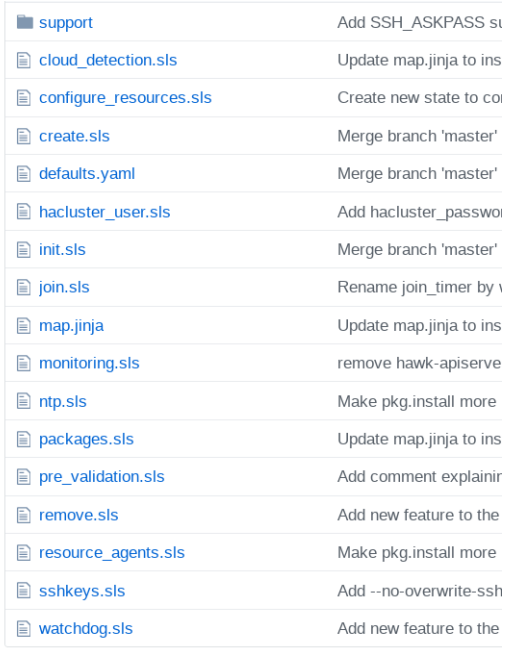
The salt states cover the initialization, joining and removal of the cluster. Initially, it will check if crmsh is available in the node to allow any operation from the crmsh salt execution module.

Most relevant states:

- **cluster_absent (remove the cluster services from the node if they are running)**
- **cluster_initialized (initialize a new cluster in this node is there is not already one running)**
- **cluster_joined (join a node to an existing cluster if the node doesn’t belong to a cluster already)**

Formulas

Salt formulas are an extensive combination of Salt states and executions around a specific concept or topic.



The formulas are a group of different sls files (yaml+jinja kind of format) which perform individual operations to gives sense to the global formula concept.

ClusterLabs HA cluster in our case!

With the formula we manage the creation, joining, removal of the nodes and pre/post installation operations (ntp, ssh keys, watchdog, etc)

Pillar and Grains files

Pillar and grains files are input data for the formulas.

- **Grains are minion specific (hostname, machine type, python version, etc) and are shared among all the salt formulas (and executions)**

- **Pillar files are specific for each formula, and they provide the input variables to make the execution customizable**

```
ec2-user@xarbulu-hana01:~> cat cluster.sls
cluster:
  name: 'hana_cluster'
  init: 'hana01'
  interface: 'eth1'
  watchdog:
    module: softdog
    device: /dev/watchdog
  sbd:
    device: '/dev/vdc'
  ntp: pool.ntp.org
  sshkeys:
    overwrite: true
    password: linux
  resource_agents:
    - SAPHanaSR
  ha_exporter: false
  configure:
    method: 'update'
    template:
      source: /usr/share/salt-formulas/states/hana/templates/scale_up_resources.j2
      parameters:
        sid: prd
        instance: "00"
        virtual_ip: 192.168.107.50
        virtual_ip_mask: 24
        prefer_takeover: true
        auto_register: false
```

But salt is only to provision the machines…
And if we get some help from other tools

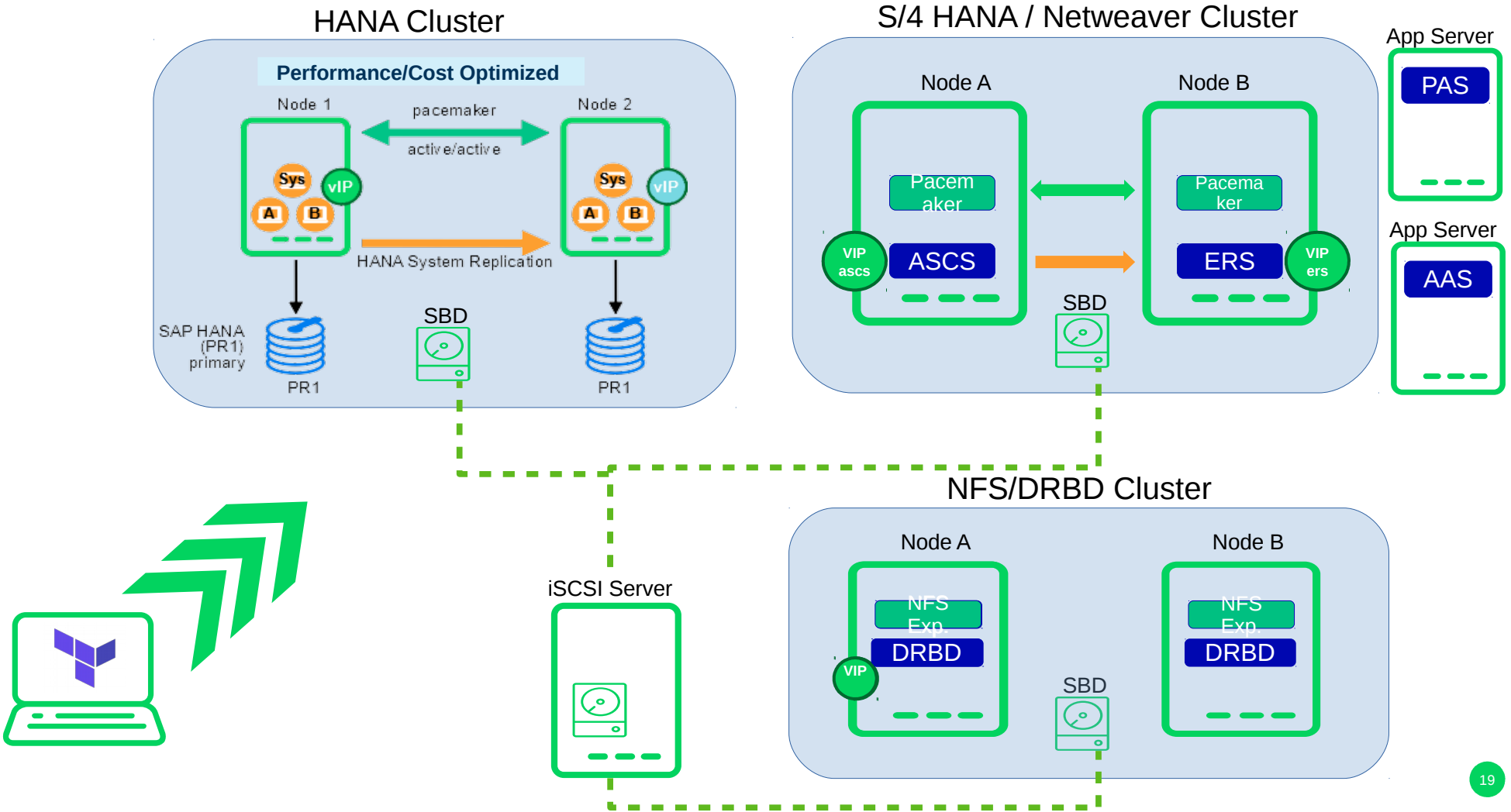
Terraform to the rescue!

- **Terraform is a tool for building, changing, and versioning infrastructure safely and efficiently**
- **Terraform can manage existing and popular service providers, like cloud, libvirt and many others**
- **Terraform can manage low-level and high-level components such as instances, storage, networking, DNS entries, etc**
- **Allow IaC, using a high-level configuration syntax. This allow versioning, sharing and reusing the infrastructure code**

Combining both, we deliver...

- Our engineering goal: Improved user experience for our cluster users 
- Fast and secure way to deploy your Cluster (DRBD, HANA, etc)
  - Minutes or hours instead of days
  - Idempotent States
- Customizable and modular “blocks”, allowing customers to reuse it and adjust for their specific needs on premises, clouds or hybrid-clouds
- It can be integrated on existing solutions like SUSE Manager, or existing Terraform and Salt, + others...

Here an example of what we can achieve



Demo \o/

Quickstart - ha-sap-terraform-deployment

In order to use the terraform project and the salt formulas (Azure will be used in this example) we need to follow the next steps (the installation of packages like terraform v0.12, git, azure-cli, etc is assumed):

Login in your azure account. The easiest way is to use tha azure-cli option.

```
az login
```

Clone the git repository and access to the folder.

```
git clone git@github.com:SUSE/ha-sap-terraform-deployments.git

cd ha-sap-terraform-deployments/azure
```

Quickstart - ha-sap-terraform-deployment

Tune your terraform.tvars file. Use the terraform.tfvars.example for reference. More details about all the parameters in the variables.tf file.

```
cp terraform.tfvars.example terraform.tfvars
```

Create SSH keys that the cluster will use to communicate between the nodes (hana, drbd and netweaver clusters will use these keys).

```
mkdir ../salt/hana_node/files/sshkeys

ssh-keygen -t rsa -f ../salt/hana_node/files/sshkeys/cluster.id_rsa
```

Quickstart - ha-sap-terraform-deployment

Create/copy the pillar files to the salt folders. The pillar files are used to configure your particular scenario. For example: how to deploy your HANA clusters (use unicast/multicast, create sbd device, set the resource agents, etc). This applies to HANA, DRBD and NETWEAVER deployments. The files need customization!

The pillar file examples are in the salt formulas projects, but some default files can be found in pillar_examples. The automatic folder stores some files configured to work in all the providers.

```
cp ../pillar_examples/automatic/hana/* ../salt/hana_node/files/pillar

cp ../pillar_examples/automatic/drbd/* ../salt/drbd_node/files/pillar
```

Quickstart - ha-sap-terraform-deployment

Run the deployment! The most convenient way for that is to create a workspace. The workspace text is used as a prefix for the most of the created resources, what makes the visualization in the webgui-s easier. And it will give the option to have several parallel deployments as well.

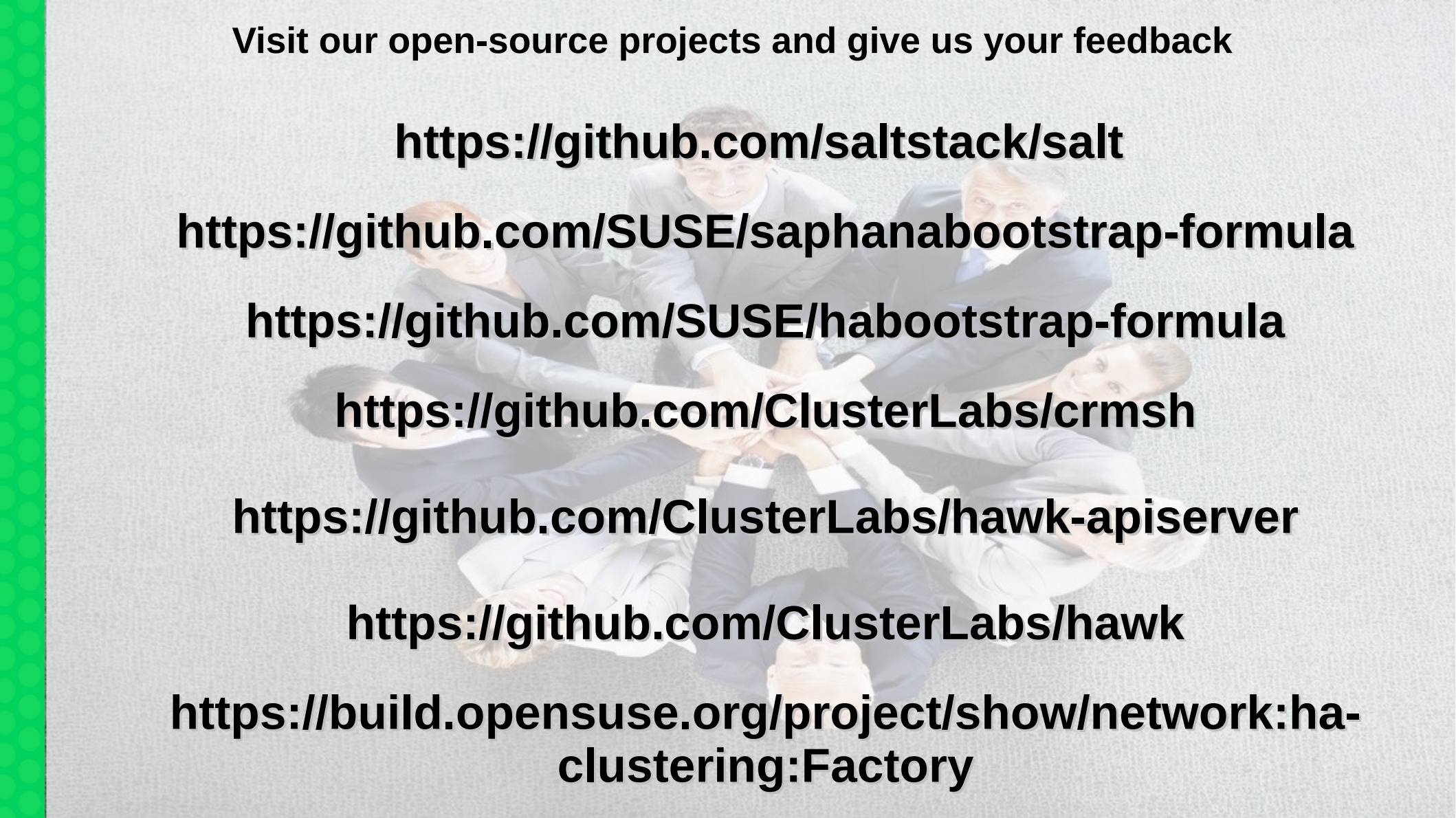
```
terraform workspace new xarbulu

terraform init

terraform plan

terraform apply -auto-approve
```

To destroy the deployment

```
terraform destroy -auto-approve
```

Our future plans…

- Continue adding more features to the salt projects
- Give more visibility to the potential use cases (big deployments, deployments in the cloud where the machines are created and destroyed periodically)
- Create custom resource agents templates in the projects to have “Ready to go” deployments for the most used use cases
- Improve the Cluster Monitoring and Troubleshoot experience



Visit our open-source projects and give us your feedback

https://github.com/saltstack/salt

https://github.com/SUSE/saphanabootstrap-formula

https://github.com/SUSE/habootstrap-formula

https://github.com/ClusterLabs/crmsh

https://github.com/ClusterLabs/hawk-apiserver

https://github.com/ClusterLabs/hawk

https://build.opensuse.org/project/show/network:ha-clustering:Factory

Thank you!


SUSE®
We adapt. You succeed.