PUPPET

Consider a system administrator working with multiple servers. If one of the servers has an issue, they can easily fix it. The situation becomes problematic, however, when multiple servers are down, This is where Puppet can help

What is Puppet?

- **Puppet** is a system management tool for centralizing and automating the configuration management process. Puppet is also used as a software deployment tool.

- It is an open-source configuration management software widely used for server configuration, management, deployment, and orchestration(Orchestration is **the automated configuration**.) of various applications and services across the whole infrastructure of an organization.

- Puppet is specially designed to manage the configuration of Linux and Windows systems. It is written in Ruby.

What is Configuration Management?

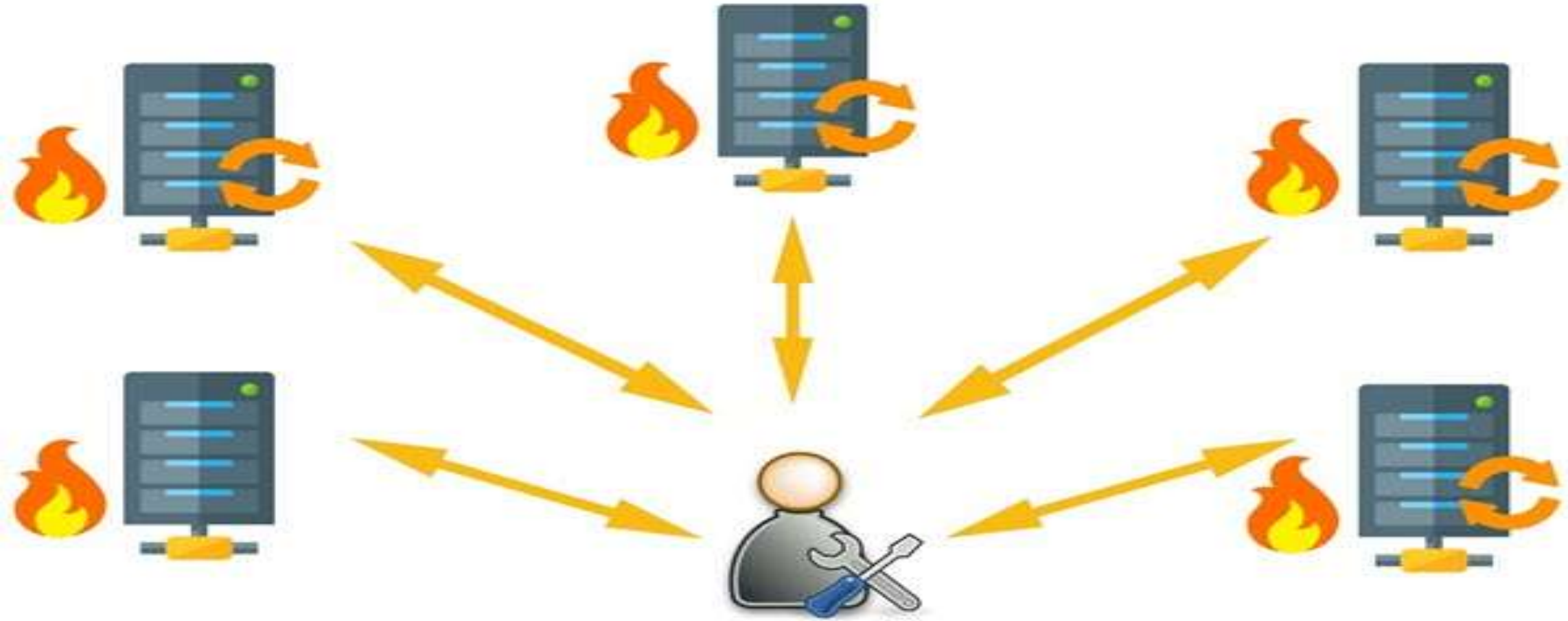
- Configuration management is the process of maintaining software and computer systems (for example servers, storage, networks) in a known, desired and consistent state. It also allows access to an accurate historical record of system state for project management and audit purposes.

- System Administrators mostly perform repetitive tasks like installing servers, configuring those servers, etc. These professionals can automate this task, by writing scripts.

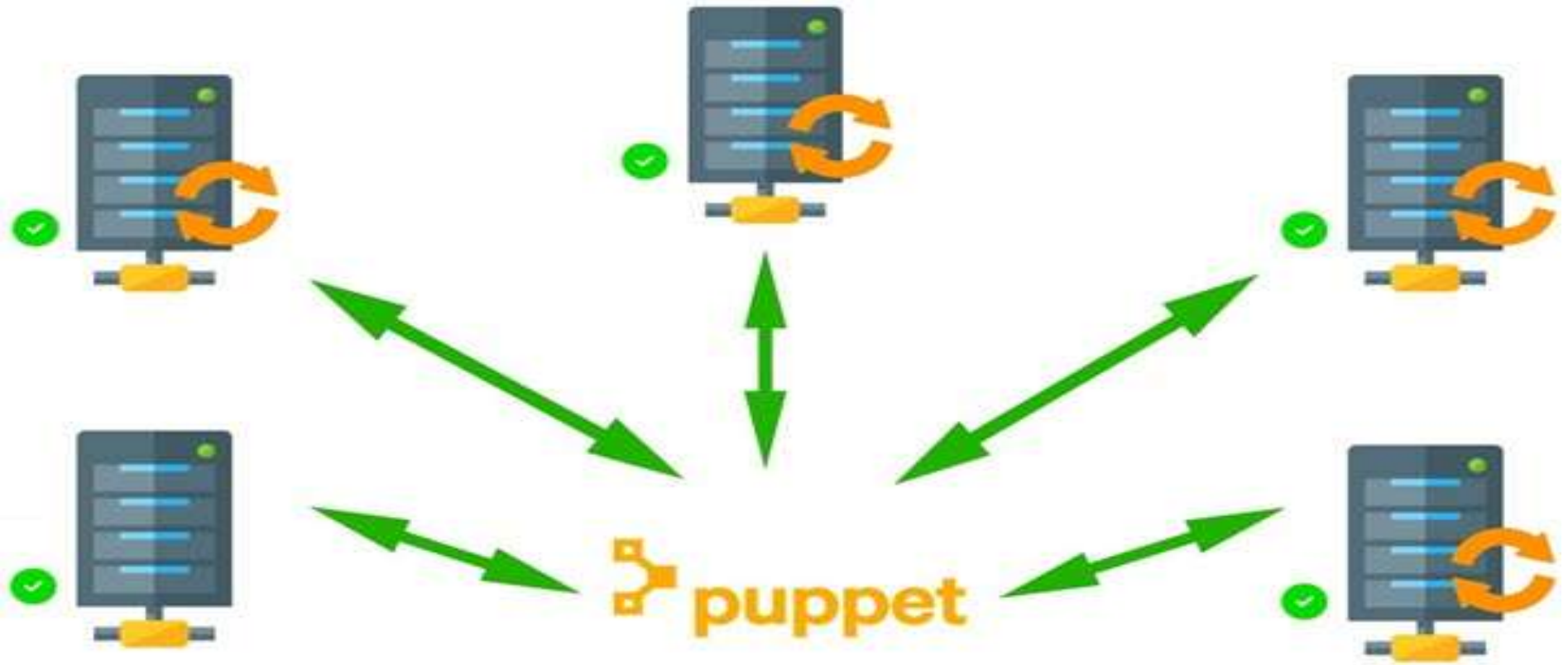
- However, it is a difficult job when they are working on a massive infrastructure. The Configuration Management tool like a Puppet was introduced to resolve such issues.

What Puppet can do?

For example, you have an infrastructure with about 100 servers. As a system admin, it’s your role to ensure that all these servers are always up to date and running with full functionality.



To do this, you can use Puppet, which allows you to write a simple code which can be deployed automatically on these servers. This reduces the human effort and makes the development process fast and effective.



Puppet performs the following functions:

- Puppet allows you to define distinct configurations for every host.
- The tool allows you to continuously monitor servers to confirm whether the required configuration exists or not and it is not altered. If the config is changed, Puppet tool will revert to the pre-defined configuration on the host.
- It also provides control over all the configured system, so a centralized change gets automatically effected.
- It is also used as a deployment tool as it automatically deploys software to the system. It implements the infrastructure as a code because policies and configurations are written as code.

Deployment models of configuration management tools

There are two deployment models for configuration management tools :

- Push-based deployment model: initiated by a master node.
- Pull-based deployment model: initiated by agents.

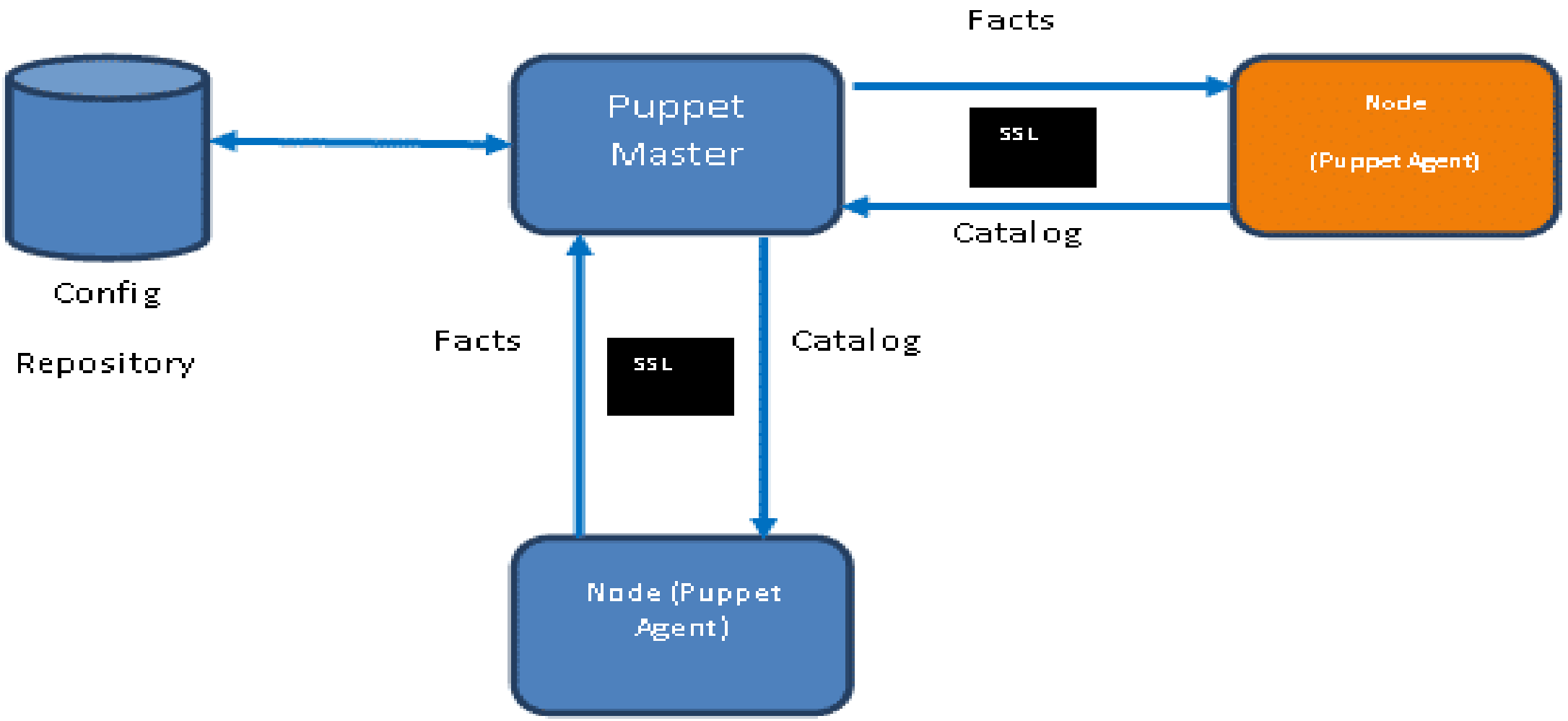
Push-based deployment model:

In this deployment model master server pushes the configurations and software to the individual agents. After verifying a secure connection, the master runs commands remotely on the agents. For example, Ansible and Salt Stack.

Pull-based deployment model.

In this deployment model, individual servers contact a master server, verify and establish a secure connection, download their configurations and software and then configure themselves accordingly — for example, Puppet and Chef.

Understanding the Puppet Architecture and Puppet Components



Client-Server Architecture

How Puppet works?

Puppet is based on a Pull deployment model, where the agent nodes check in regularly after every **1800** seconds with the master node to see if anything needs to be updated in the agent. If anything needs to be updated the agent pulls the necessary puppet codes from the master and performs required actions.

Example: Master – Agent Setup:

The Master:

A Linux based machine with Puppet master software installed on it. It is responsible for maintaining configurations in the form of puppet codes. The master node can only be Linux.

The Agents:

The target machines managed by a puppet with the puppet agent software installed on them.

The agent can be configured on any supported operating system such as Linux or Windows or Solaris or Mac OS.

The communication between master and agent is established through secure certificates.

•**Config repository:** Config repository is where the entire server-related configurations and nodes are stored. They can be pulled at any time as required.

Catalog

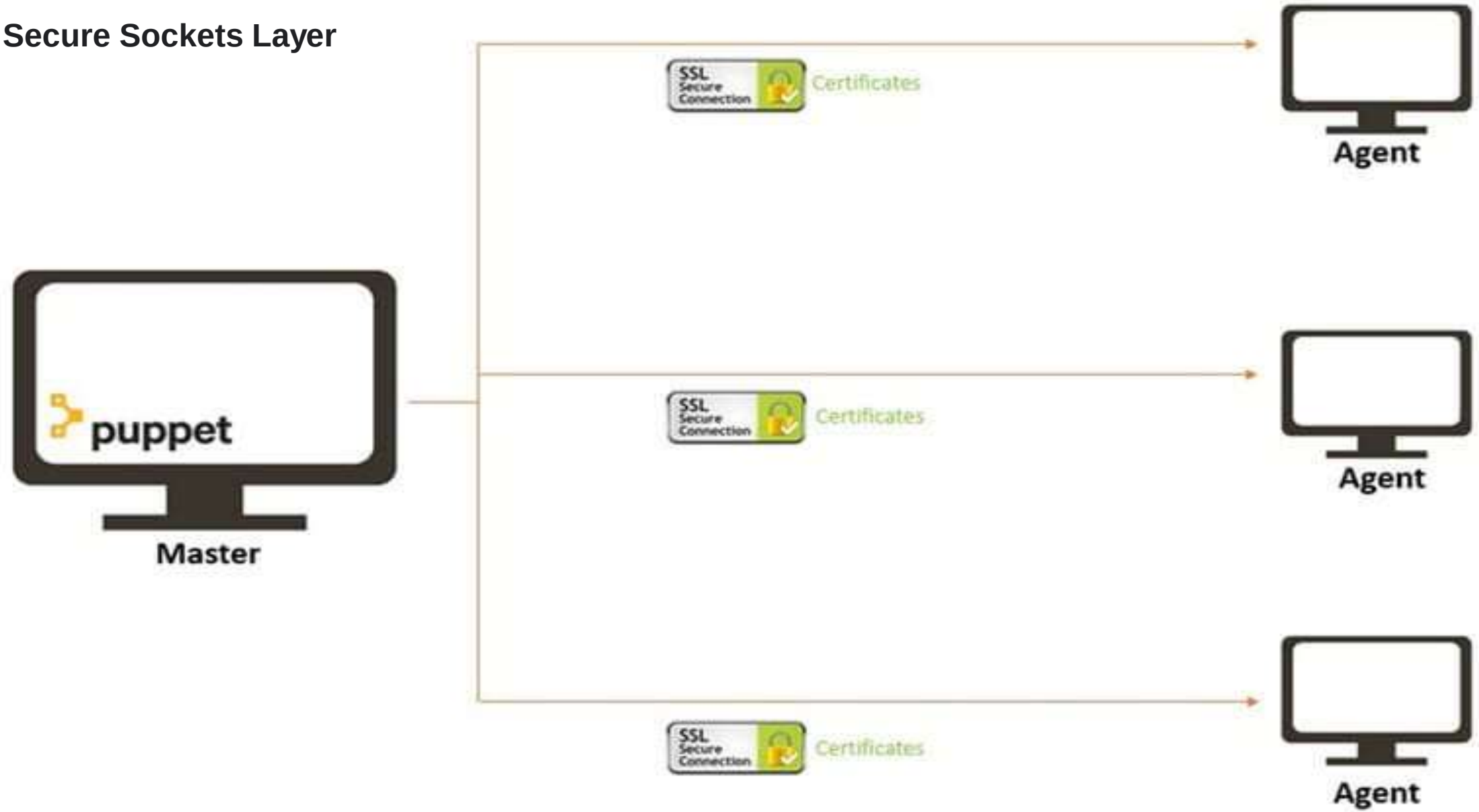
A catalog is a document that describes the desired system state for one specific server. It lists all of the resources that need to be managed, as well as any dependencies between those resources.

Manifests

Manifests are files with extension ".pp", where we declare all resources to be checked or to be changed. Resources may be files, packages, services and so on.

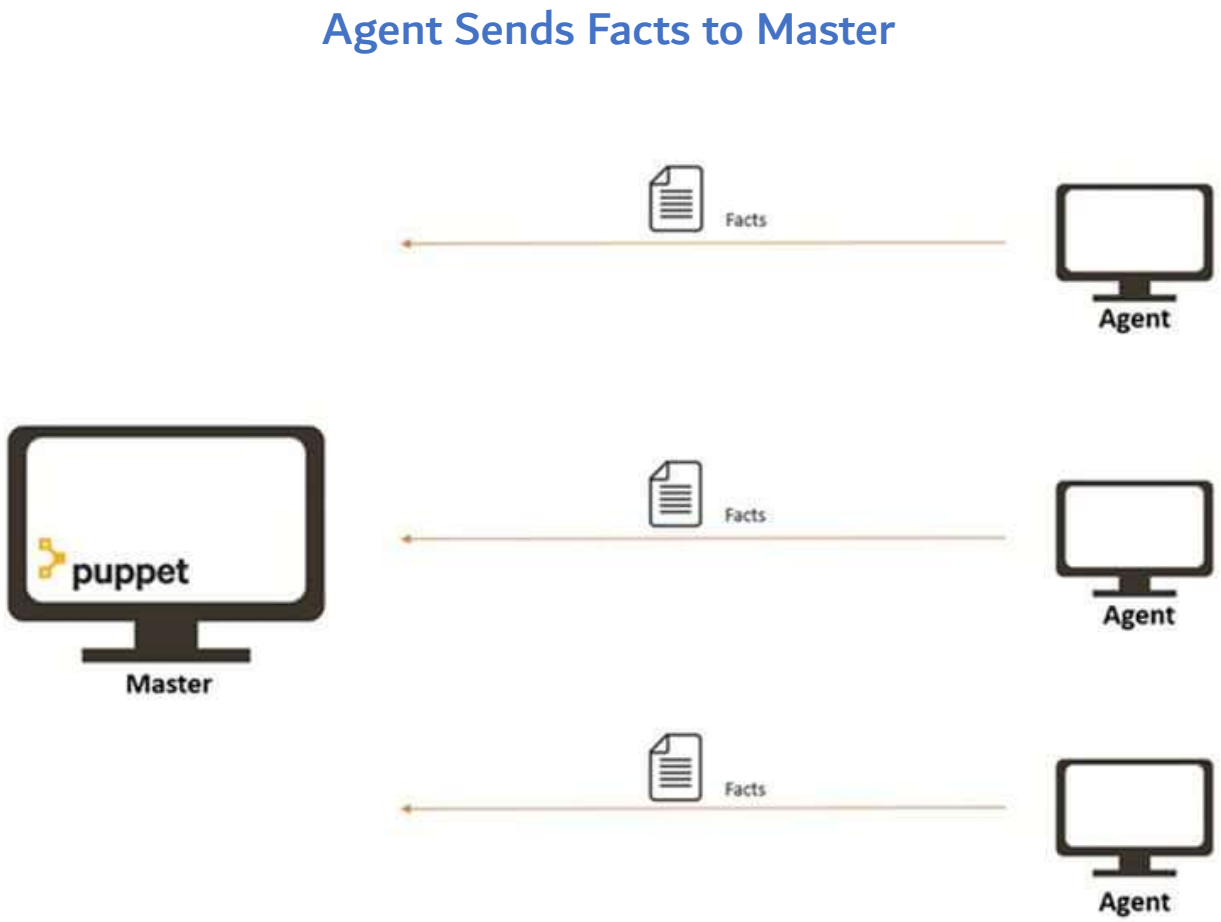
Puppet Master Agent Communication

Secure Sockets Layer



Communication between the Master and the Agent:

Step 1) Once the connectivity is established between the agent and the master, the Puppet agent sends the data about its state to the Puppet master server. These are called Facts: This information includes the hostname, kernel details, IP address, file name details, etc.…



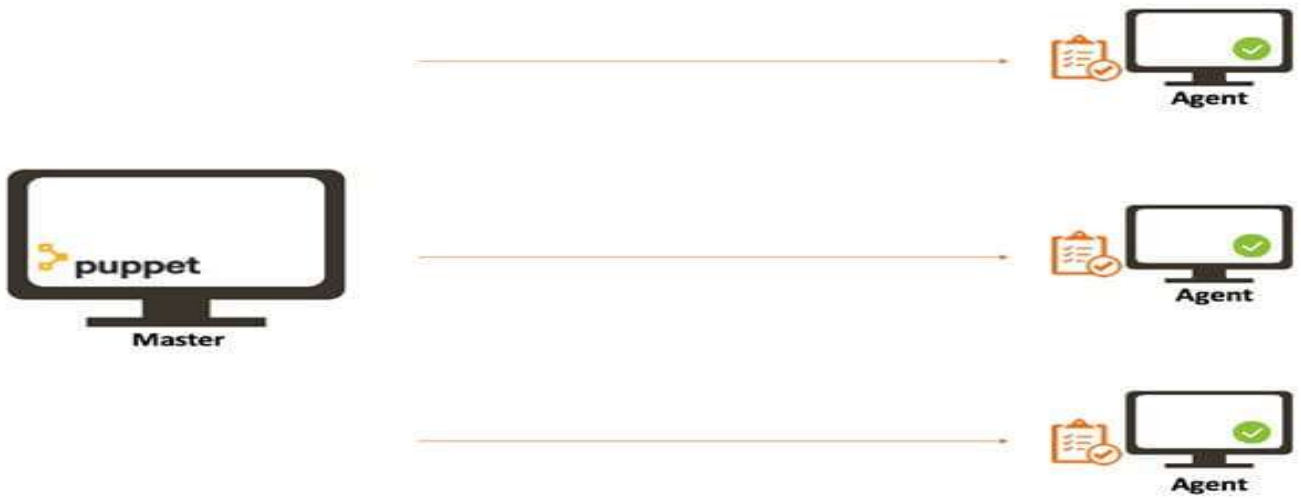
Step 2) Puppet Master uses this data and compiles a list with the configuration to be applied to the agent. This list of configuration to be performed on an agent is known as a **catalog.** This could be changed such as package installation, upgrades or removals, File System creation, user creation or deletion, server reboot, IP configuration changes, etc.

Master sends a catalog to Agent

Step 3) The agent uses this list of configuration to apply any required configuration changes on the node.

In case there are no drifts in the configuration, Agent does not perform any configuration changes and leaves the node to run with the same configuration.

Agent applies configuration



Step 4) Once it is done the node reports back to puppet master indicating that the configuration has been applied and completed.

The Connection Between Puppet Master Server and Puppet Agent Nodes

As we can see, the Puppet master and the Puppet agents have to communicate with each other in order to let Puppet work seamlessly, but how exactly do they communicate?

Let’s understand the workflow of Puppet step by step:

•The nodes that the Puppet master controls have Puppet agents installed on them. The agents collect all the configuration information about their particular nodes using facts. Agents then send facts to the Puppet master.

•After gaining all the information, the Puppet master compiles a catalog based on how the nodes should be configured. The Puppet master then sends back the catalog to the agents.

•Each agent uses these catalogs and the information in them to make necessary configuration updates on their nodes and then reports back to the Puppet master.

•The Puppet master can also share the reports with a third-party tool if needed.

•The Puppet master communicates with a Puppet agent via HTTPS (HyperText Transfer Protocol Secure) with client verification.

•The Puppet master provides an HTTP interface. Whenever the Puppet agent has to make a request or submission to the Puppet master,

•it just makes an HTTPS request to one of the endpoints available in the HTTP interface provided by the Puppet master.


Puppet Agent
Request for Master Certification
Send Master Certificate
Request for slave certificate
Send Slave Certificate
Request for Data
Send Data
Puppet Master


ANSIBLE

What is Ansible?(Push based)

- Ansible is an open source DevOps tool which can help the business in configuration management, deployment, provisioning(**the process of setting up IT infrastructure**), etc.

- It leverages SSH(Secure Shell is **a network communication protocol that enables two com**(http or hypertext transfer protocol, which is the protocol used to transfer hypertext such as web pages) **to communicate..** and share data to communicate between servers.

- Ansible provides reliability, consistency, and scalability to your IT infrastructure. You can automate configurations of databases, storage, networks, firewalls using Ansible.

- It makes sure that all the necessary packages and all other software are consistent on the server to run the application.

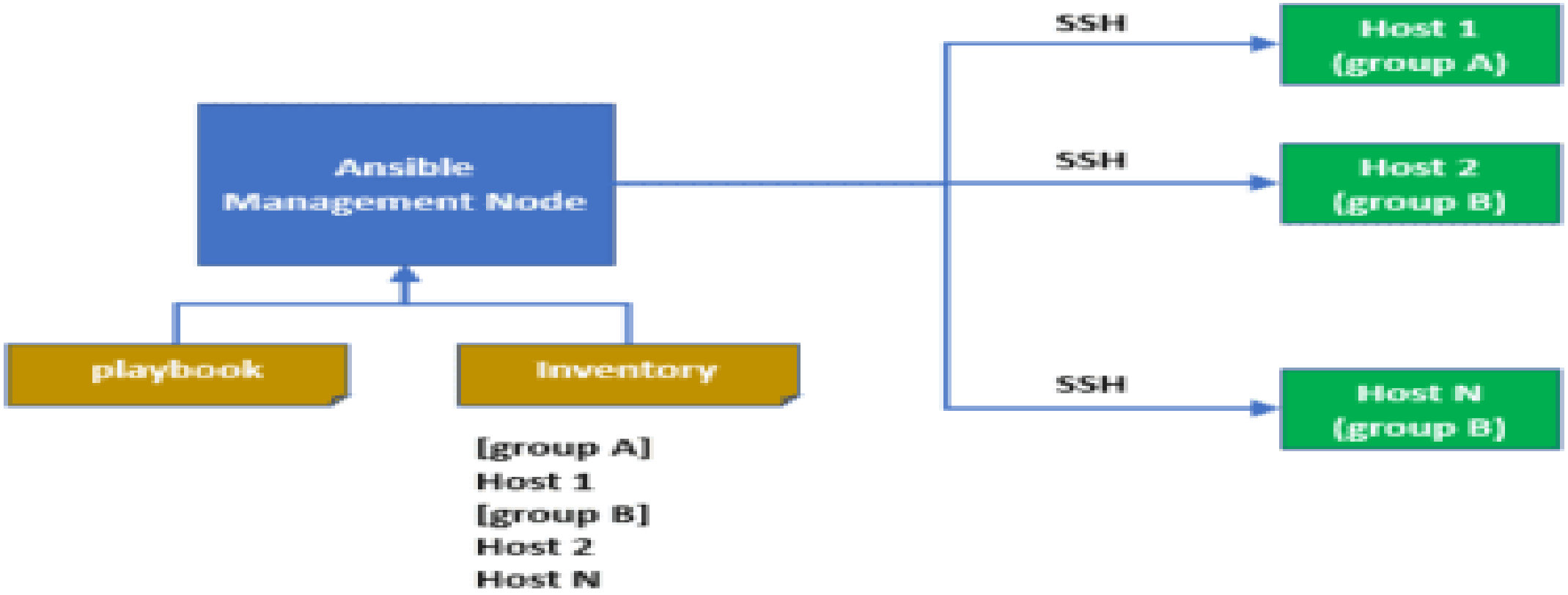
Let’s take an example; you’ve got a debug version of an application that is built on visual C++.

Now if you want to run that application on a computer, you would need to meet some prerequisites like Microsoft Visual C++ library DLLs, and you would need visual C++ installed in your computer.

So, this is the part where Ansible will make sure that all these basic packages and all the software’s are installed in your computer so that your application can run smoothly on all the environments, may it be test or production environment.

It also holds all the historical data of your application, so if at any time you want to roll back to the previous version, or you want to upgrade it, you can easily do that.

How Ansible Works?



What are playbook tasks?

A task is **the smallest unit of action you can automate using an** Ansible playbook. Playbooks typically contain a series of tasks that serve a goal, such as to set up a web server, or to deploy an application to remote environments. Ansible executes tasks in the same order they are defined inside a playbook.

Ansible Workflow

Ansible works by connecting to your nodes and pushing out a small program called **Ansible modules** to them.

Then Ansible executed these modules and removed them after finished. The library of modules can reside on any machine.

In the above image, the **Management Node** is the controlling node that controls the entire execution of the playbook.

The **inventory** file provides the list of hosts where the Ansible modules need to be run.

The **Management Node** makes an **SSH** connection and executes the small modules on the host's machine and install the software.

Ansible removes the modules once those are installed so expertly. It connects to the host machine executes the instructions, and if it is successfully installed, then remove that code in which one was copied on the host machine.

Let’s take a look at some of the following features.

Agentless – Which means there is no kind of software or any agent managing the node like other solution such as puppet and chef.

Python – Built on top of python, which is fast and one of the robust programming languages in today’s world.

SSH – Very simple password less network authentication protocol which is secure. So, your responsibility is to copy this key to the client

Push architecture – Push the necessary configurations to them, clients. All you have to do is, write down those configurations (playbook) and push them all at once to the nodes. You see how powerful it can be to push the changes to thousands of servers in minutes.

Setup – a minimal requirement and configuration needed to get it to work.

Benefits of Ansible

•Free: Ansible is an open-source tool.

•Very simple to set up and use: No special coding skills are necessary to use Ansible's playbooks (more on playbooks later).

•Powerful: Ansible lets you model even highly complex IT workflows.

•Flexible: You can orchestrate the entire application environment no matter where it's deployed. You can also customize it based on your needs.

•Agentless: You don't need to install any other software or firewall ports on the client systems you want to automate. You also don't have to set up a separate management structure.

•Efficient: Because you don't need to install any extra software, there's more room for application resources on your server.

Next, in our path to understanding what ansible is, let us find out the features and capabilities of Ansible.

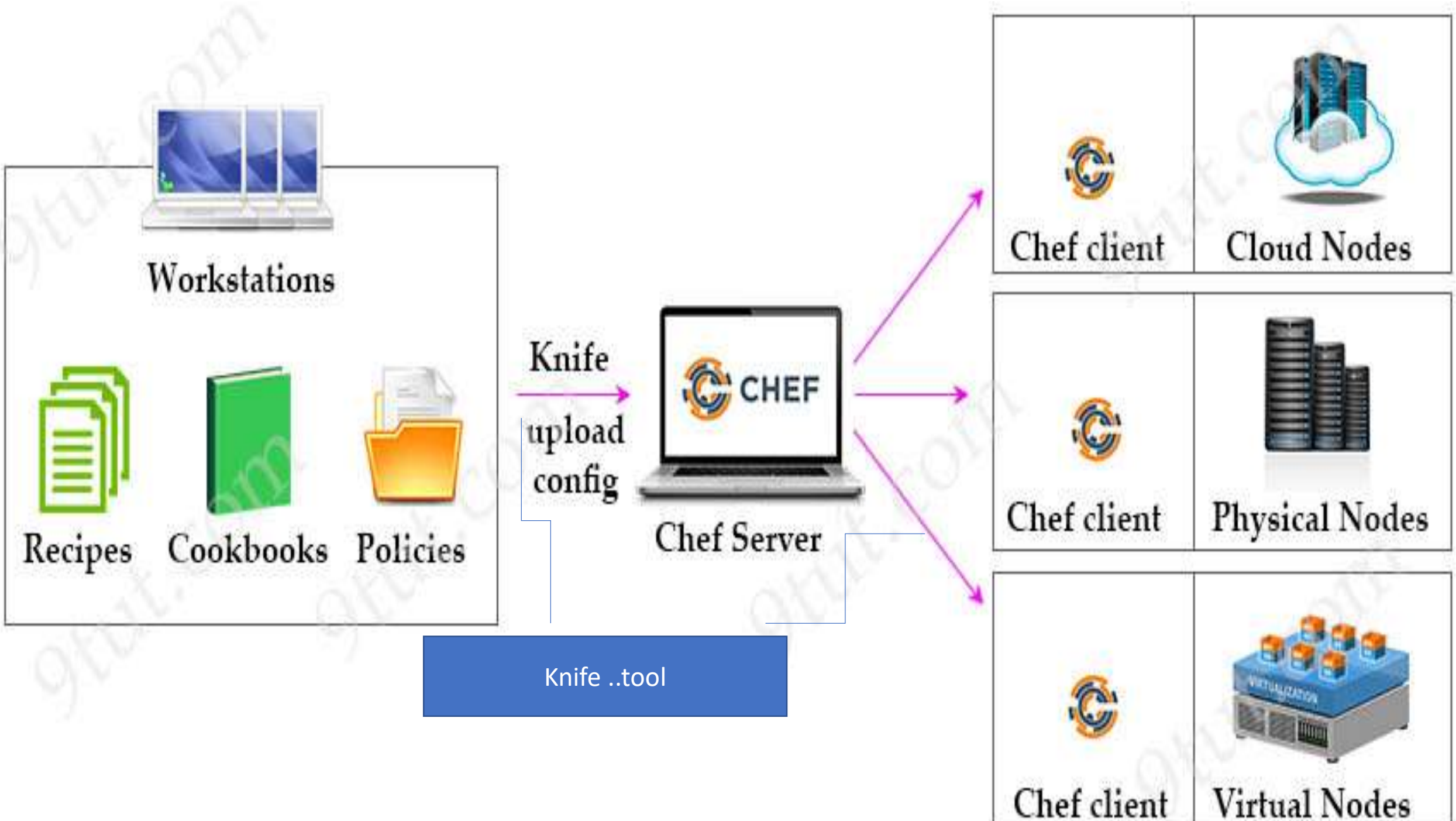
CHEF

Chef is an automation platform that configures and manages your network infrastructure.(**PULL BASED**) Chief transforms infrastructure into code. “Infrastructure into code” here means “deploy your code/application/configuration and policy” on many machines or instances automatically via your code.

Chef is a tool used for Configuration Management which closely competes with Puppet. Chef is an automation tool that provides a way to define infrastructure as code.

As shown in the diagram below, there are three major Chef components:

- Workstation
- Server
- Nodes


Workstations
Recipes
Cookbooks
Policies
Knife
upload
config
CHEF
Chef Server
Chef client
Cloud Nodes
Chef client
Physical Nodes
Chef client
Virtual Nodes
Knife ..tool

Workstations: simply personal computers where all development configuration code is created, tested, and changed before uploading to the Chef Server.

Each Chef workstation also has a command line tool called "Knife", which will be used to upload configuration changes to the Chef Server.

Workstations are the place to write Recipes and Cookbooks:

- Recipes: A Recipe is a collection of resources that describes a particular configuration ,how to implement,code.
- Cookbooks: Multiple Recipes can be grouped together to form a Cookbook. A Cookbook defines a scenario and contains everything that is required to support that scenario.

- policy. It describes everything that is required to configure part of a system and in which order it is to be used. The user writes Recipes that describe

- how Chef manages applications and utilities (such as Apache HTTP Server, MySQL, or

- Cookbooks: Multiple Recipes can be grouped together to form a Cookbook. A Cookbook defines a scenario and contains everything that is required to support that scenario.

- A Cookbook also includes attributes, libraries, metadata, and other files that are necessary for supporting each configuration.

- Cookbooks are created using Ruby language is used for specific resources.

- **Writing Cookbooks and Recipes that will later be pushed to the central Chef Server**

- **Chef Server**: The centralized store of our infrastructure's configuration. The Chef server stores, manages and provides configuration to all nodes that make up the infrastructure.

- **Nodes**: are the servers where your code needs to run. Chef server manages Nodes by **Chef client**, which is a software installed on each Node.

- Chef client retrieving configuration information from the Chef Server. Nodes can be a cloud-based/virtual/physical server in your own data center.

Any changes made to your infrastructure code must pass through the Chef server in order to be applied to nodes. Prior to accepting or pushing changes, the Chef server authenticates all communication via its REST API using public key encryption.

Chef client periodically pulls Chef server to see if there are any changes in cookbooks or settings. If there are changes then Chef server sends the latest configuration information to Chef client. Chef client applies these changes to nodes.


1 Upload configs
2 Check if config changes
3 If yes, sends new config
4 Update Node
CHEF
Chef client
Workstations
Chef Server
Node

SALTSTAKER

Why Saltstack?

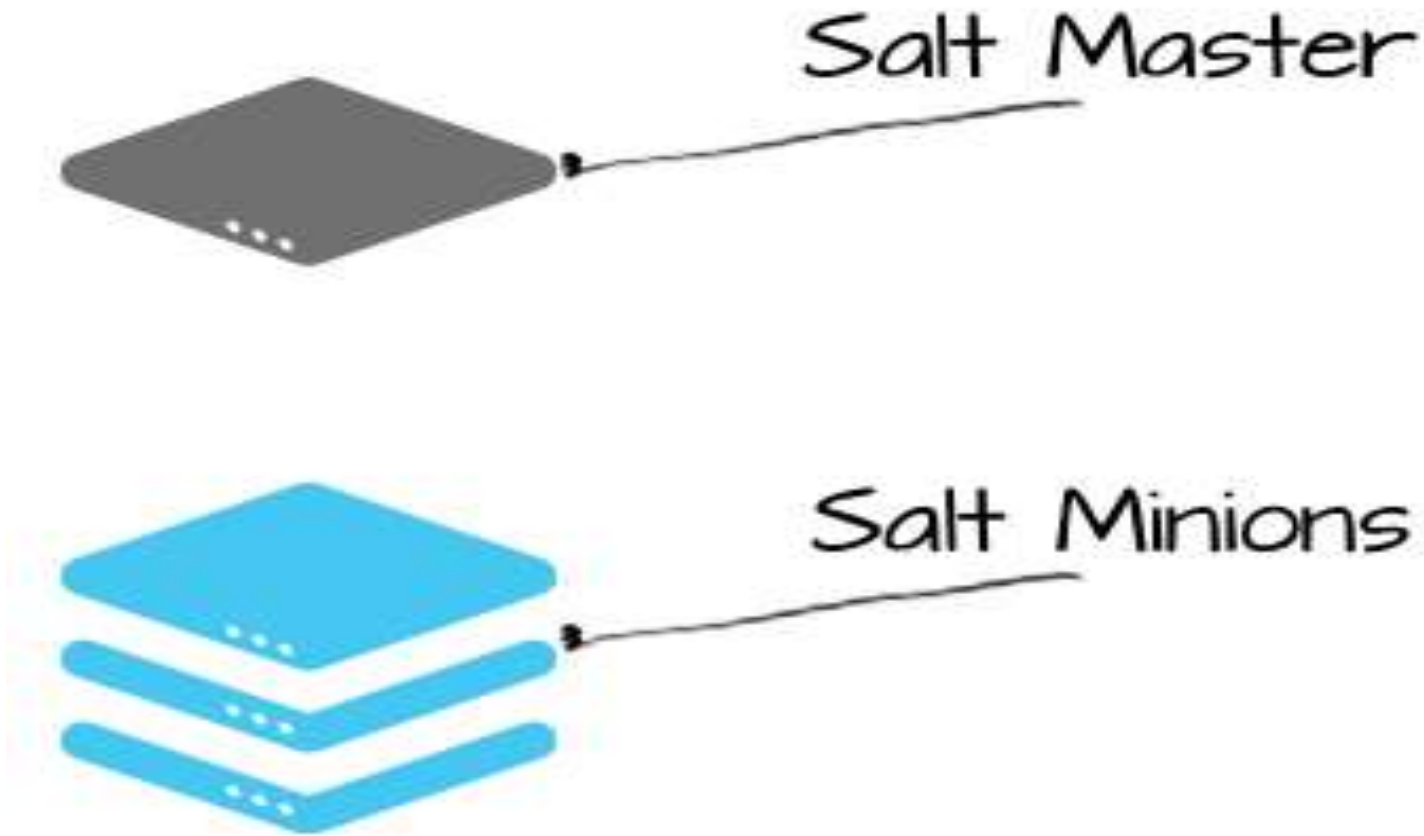
SaltStack Architecture is developed for providing speed and scale.

This is the reason why this architecture is used for managing the thousands of servers at Google, Linked In, and many more companies.

For example - If a user has thousands of servers and the user wants to perform functions on every server. A user would be required to log in to every server and perform functions one at a time. This process will be very complicated and time-consuming as functions like software installation and configuration based on particular criteria may consume a lot of time.

To overcome this problem, SaltStack Architecture has been introduced as one can perform different functions just by typing one command. Thus, SaltStack is a single solution to overcome all such issues

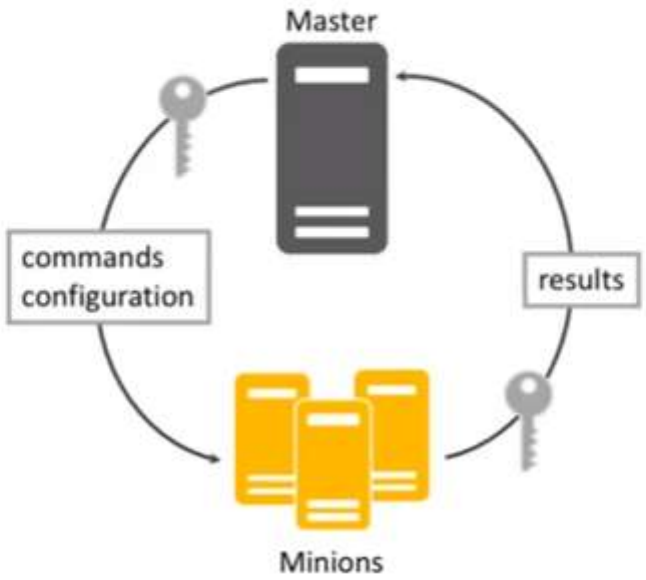
Salt uses a server-agent communication model, The server component is called the Salt master, and the agent is called the Salt minion.



The Salt master is responsible for sending commands to Salt minions, and then aggregating and displaying the results of those commands. A single Salt master can manage thousands of systems.

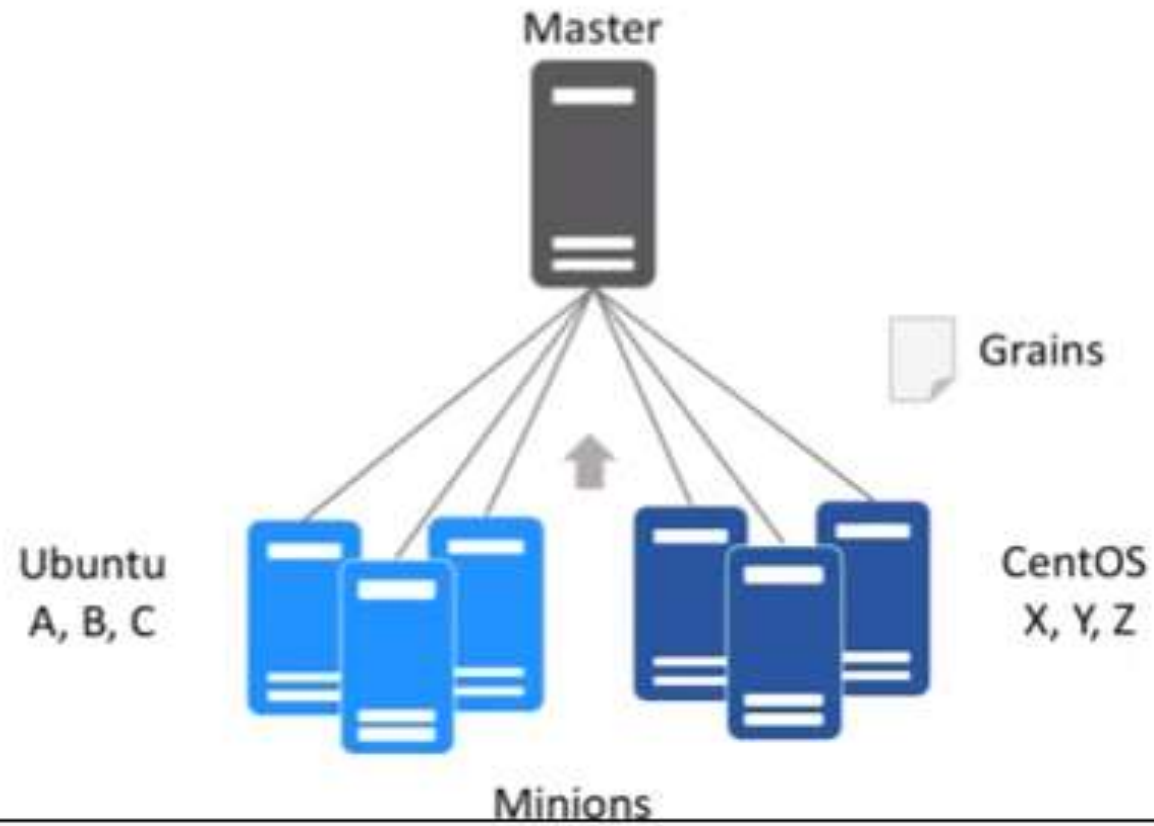
SALTSTACK ARCHITECTURE

- Client server model.



SALT GRAINS

- Static information about the minion
- Eg: OS, memory, model, serial number etc.



Salt is a **simple IT automation framework** that automates provisioning(Providing), configuration management, application deployment, service arrangements and many other IT needs.

✓ Remote execution of commands is the important highlight of using Salt.

✓ Unlike other competitors, Salt uses **ZeroMQ message bus** for the communication.
{ ZeroMQ API provides sockets , each of which can represent a many-to-many connection between endpoints. }

✓ It offers high speed effective communication between remote machines.

✓ Additionally, the remote execution can target a group of machines simultaneously.

The relevance of Salt and its applications

- ✓ Salt (commonly known as SaltStack) is a widely accepted tool for Configuration management.

- ✓ Consider that hundreds of machines need to be configured identically - install software, run some services, and make it more awesome.

- ✓ Rather than performing these tasks manually, scripts can be used to simplify it. Since the same task needs to be executed in hundreds of machines, if the script is placed in a location that can be send to all these machines will further simplify the repetitive task.

- ✓ With the help of Salt, we can manage this activity from a single system (named Salt master) and complete this in a swift manner. Just write the task once, specify the machines to be modified and execute the script remotely. The result from all machines will be send back to Salt master.

Features of Salt

1. Simplicity - The Salt setup and usage is very simple process. Still it is powerful enough to manage hundreds of distributed systems in quick and efficient manner

2.Parallel Execution - The Salt remote execution is effected in nodes parallelly, not serially. Hence, the time taken will be very less

3. Open Source - It is a open source tool

4. Fast, Flexible, Scalable - By making use of ZeroMQ message bus, the configuration management will be fast and it supports various types of models like master and minion, master-only, minion-only or combination of these models. The support to include ten thousands of minions per single master displays the effectiveness of scalability

5. Simple Interfaces - Salt can be accessed from the command line, as well as from any simple Python API

Salt Architecture:

Salt remote execution is built on top of an event bus unlike other configuration management tools. Event bus makes communication between senders and receivers simple and fast.

It is specifically optimized for high performance. In Salt , the salt master is a server where the salt-master service is running. It issues commands to salt-minions, that are servers where the salt-minion service is running.

Public keys are used for authentication with master daemon, for communication it uses faster AES (Advanced Encryption Standard) encryption.

After the execution of job in minion, the job return data is sent back to master.

By default two ports are used by Salt, for the minions to communicate with master.

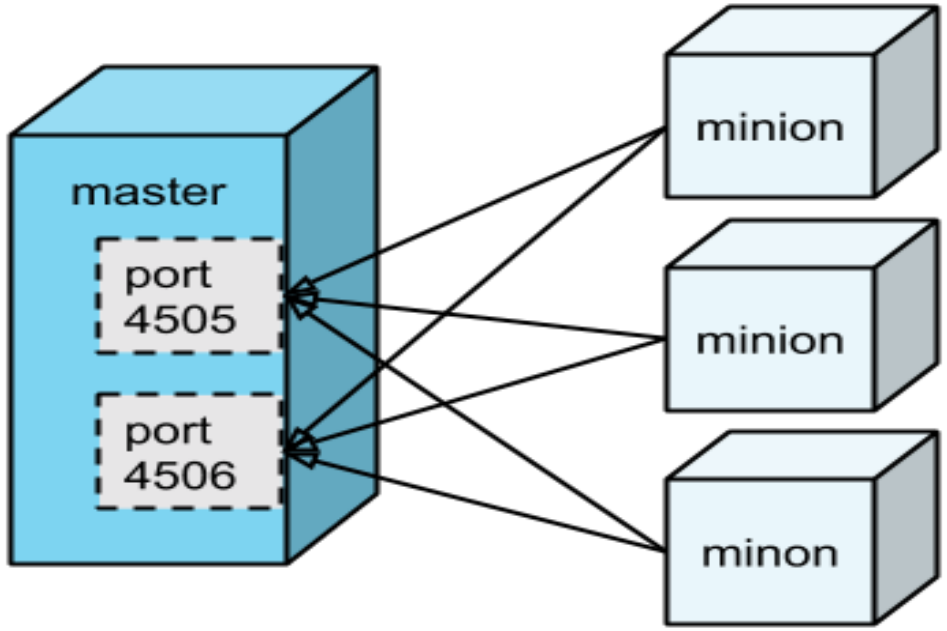
These ports work together to receive and send data to the Message Bus.

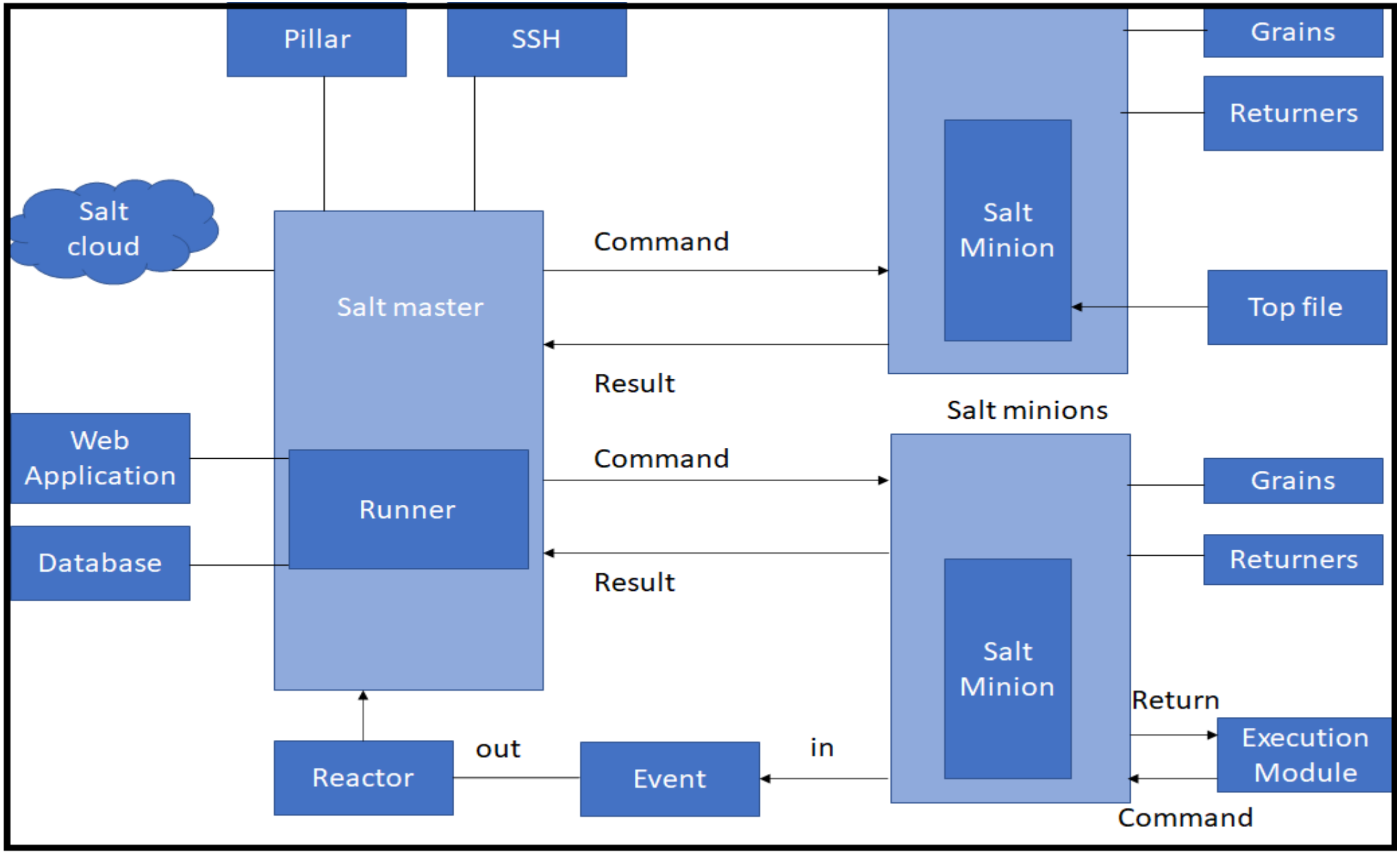
Salt uses ZeroMQ message bus. ZeroMQ provides the fastest communication possible.

Salt minions are responsible for connecting to the Salt master. Minions receive *jobs* (i.e., instructions) via the open connection to port 4505, and send results (i.e., status updates) via the open connection to master 4506.

That is, Salt masters publish jobs to all connected minions, and it is the responsibility of each minion itself to determine whether or not it is an intended *target* (i.e., whether or not to run the job).

Communication between master and minions is done using ZeroMQ, an asynchronous messaging library.




Pillar
SSH
Grains
Returners
Salt cloud
Salt master
Command
Salt Minion
Top file
Result
Salt minions
Web Application
Runner
Command
Grains
Database
Result
Returners
Salt Minion
Return
Execution Module
Reactor
out
Event
in
Command

Salt-master - The central management system. It is used to send commands and configurations to the salt-minion that is running on managed nodes.

Salt-minion - The managed system which receives commands and configuration from the salt-master.

Execution Modules - Ad hoc commands executed from the master's command line that reflect on the minions. It performs real-time monitoring.

Formulas - A declarative or imperative representation of a system configuration. They are used for tasks such as installing a package, setting up users or permissions, configuring and starting a service and other common tasks.

Grains - Grains provides information specific to a minion. It includes information like operating system, memory, and other system properties. Grains get loaded when the salt-minion starts. Grains are system variables.

Pillar - They are user-defined variables. These variables store highly sensitive data specific to a particular minion, such as passwords and cryptographic keys. They are stored on salt-master and then using targets its assigned to one or more minions.

Top file - It contains the mapping between groups of machines on the network and the configuration roles.

Runners – It is a module within salt-master that executes to perform supporting tasks. Salt runners reports job status read data from external APIs, connection status, query connected salt-minions, and much more.

Returners − Returns data from salt-minions to another system like a database. It can run on salt-minion or salt-master.

Reactor – It triggers reactions when events occur in SaltStack environment.

SaltCloud − Provision systems on cloud providers / hypervisors and bring them under management of salt-master.

SaltSSH − Run Salt commands over SSH on the systems not having salt-minion.