Monolithic PAD

minor change, such as fixing a spelling mistake in the static website, the entire web application archive needs to be rebuilt and redeployed.

Including the possibility of revenue loss if something breaks during deployment.

making involves creating a new branch in the code base, making the correction, building a new artifact, assigning a new version, and deploying it to production.

introduce uncertainties and complexities that demand careful consideration

MAYBE **MANUAL CORR** FOR MAJOR CHANGES

Architecture rules of thumb

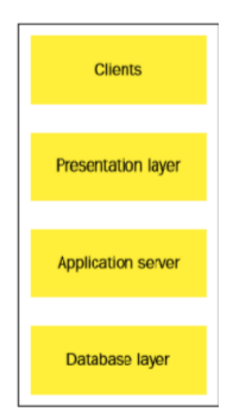
HIGH COHESION( elem of a sw mod belong 2 each other) LOW COUPLING(dependency)

Systems with high cohesion and low coupling would automatically have separation of concerns, and vice versa.

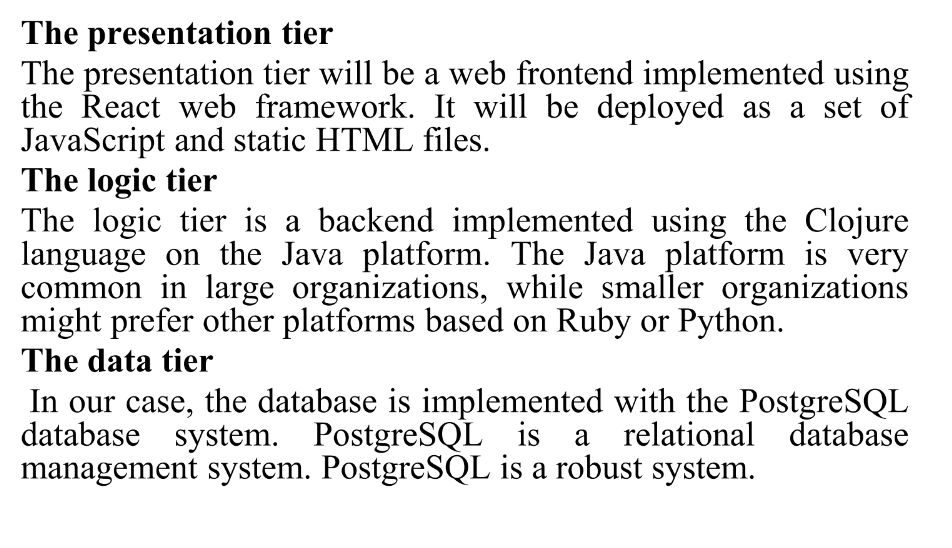
Separation of concerns

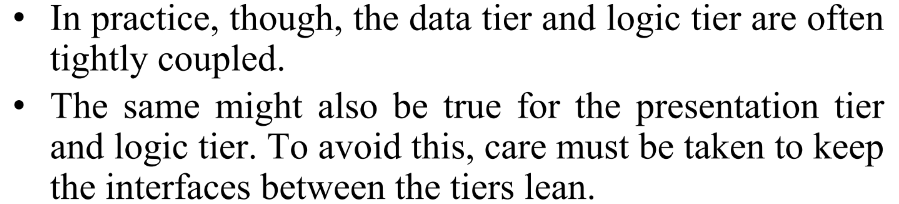
advocates breaking a computer program into distinct and independent sections.

creates of more scalable, modular, and maintainable software systems.



3tier



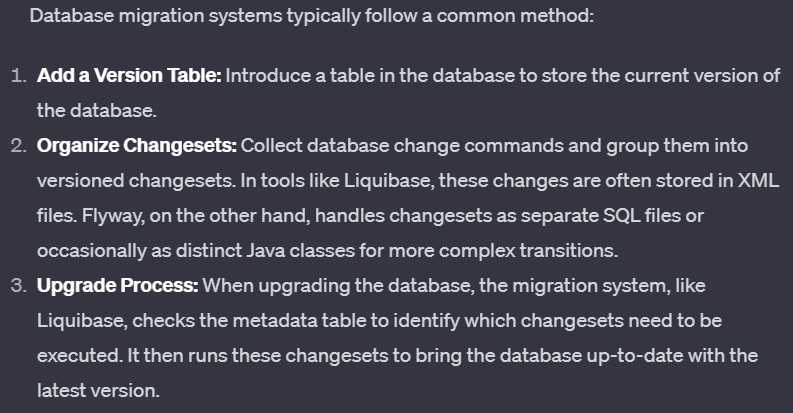
d-l , p-l tightly coupled

Handling db migrations (bin,ext,db,state,vcontrol,liquibase,)

* In rdb both data and its structure are stored, making it little complex while upgrading.
* **Unlike** upgrading app **binaries**, where the app **state** is typically **handled externally** upgrading db requires consideration of the db state.
* the process of handling database migrations involves treating the database structures before and after a change as different versions.
* Version control is imp.
* using tools like Liquibase, which is a database migration system .
* automate and streamline the process of applying database changes, ensuring consistency and version control in database structures.

A binary file is a file whose content must be interpreted by a program or a hardware

processor that understands in advance exactly how it is formatted.



Rolling upgrades

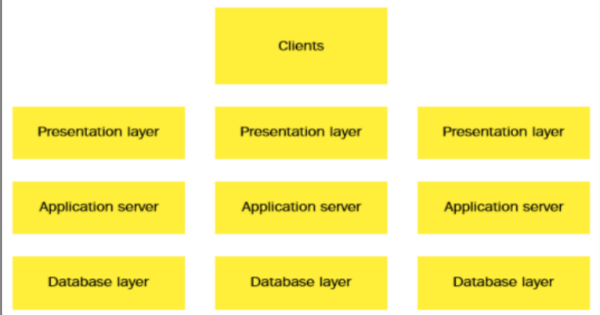
when you d**on't wa**nt your end **user to experience any downtim**e, or at least very little downtime.

example of a rolling upgrade for our organization's customer database. When we start, we have a running system with one database and **two servers**. We have a load balancer in front of the two servers. We are going to roll out a change to the database schema, which also affects the servers.

You start by **upgrading one server at a time, a**llowing the other server to continue handling requests. This ensures that there is no downtime for users during the upgrade process.

Microservices

**term used to describe systems where the logic tier of the three-tier pattern is composed of several smaller services that communicate with language-agnostic protocols.**

**Language-agnostic protocols** refer to communication standards or rules that are designed to be independent of any specific programming language. facilitate interoperability between heterogeneous systems.

architectural design pattern lends itself well to a Continuous Delivery approach easier to deploy a set of smaller standalone services than a monolith.

HTTP based, commonly JSON REST, but this is not mandatory.

DevOps, architecture, and resilience

intersection of DevOps, architecture, and resilience, particularly in the context of microservices

Challenges of Microservices:

Microservices **introduce more integration points** between systems, increasing the potential for failure compared to monolithic systems.

**Automated testing is crucial** for ensuring the quality of deployed changes but doesn't completely address the challenge of unexpected service failures.

Reseilence is recovery , ability to recover quickly and effectively from failures, disruptions, or adversities.

Resilience in Microservices Architecture:

Monitoring Strategy: (server is found to be non-functional, the load balancer is reconfigured to exclude the problematic server from the pool.)

Application-Specific Monitoring Interface

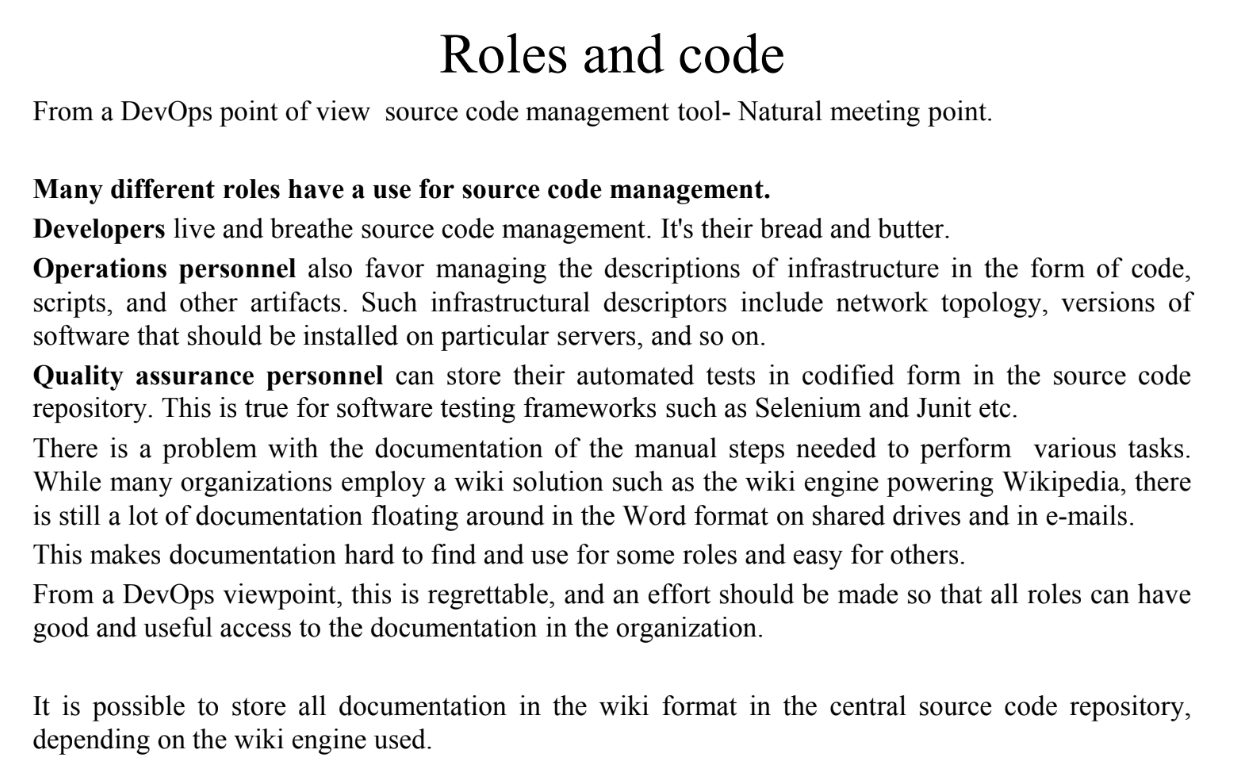
We have more running services with the microservice pattern, it is statistically more likely for a service to fail.

need for source code control

everythings code

The applications that we build

• The infrastructure that hosts our applications

• The documentation that documents our products

source code management system

many source code management (SCM) systems dominant is git

Unlike older version control systems that are centralized, where there is a single repository that everyone interacts with, DVCS allows every user to have their own local repository.

Dvcs benifiits

Distributed hence not connected to any nw

Therefore faster

Work pvtly until showable

several remote logins simultaneously, hence single pt failure not poss

others: bazaar, mercurial

Source code management system migrations

Save history if u can, however expensive



Deployment

code has been built and tested, we need to deploy it to our servers so that our customers can use the newly developed features!

Tools;

– Puppet

– Ansible

– Salt

– PalletOps etc.

1. Focus on installing binary packages and their configuration with config ms on servers
2. 3 components PAD
3. If one single physical server , download / upgrade manually even if it is err prone but best in cost effectivess
4. more likely that a large organization has hundreds of servers and applications and that

they are all deployed differently, with different requirements.

We have different solutions for that

1. unit that will exec our code can be physical server,vm,container,combi

configuring the base os

* java,ruby/py they have cross platform functionalities hence go to a gret len and have **less req**
* **os req are apprent** if u work with low level mixed hw and sw intigrations(telecom)
* Some systems work with a bare metal appr, install the os from stracth then install base dependencies that orgs needs
* Red Hat Satellite andCobbler, which works in a similar way but is more
* lightweight. Cobbler allows you to boot a physical or virtual machinever the network using dhcpd.
* Another method that is very popular today is to provide base operating system images that can be reused between machines.
* providing base operating system images that can be reused across multiple machines. Cc
* in cc new is made using pre existing
* Docker also work in a similar way, where you declare your base container image and then describe the changes you want to formulate for your own image.

Describing clusters (multiple machines, or nodes, collaborate)

* If single machine no use of cluster deployment.
* reality is normally that your applications are spread out over a set of machines, virtual or physical.
* normally also used to describe the application layer.
* Much of an app can be installed as packages, which are installed unmodified on the target sys by the cms
* RPM and deb have useful features, such as verifying that the files provided by a package are not tampered with on a target system, by providing checksums for all files in the package.
* Puppet has an extensive system that allows machines to have different roles that in turn

imply a set of packages and configurations.

Delivering packages to a system

* relies on text-based configuration files. (usually) can do by manage them in source code handling systems such as Git.
* other method application that provides an API to handle configuration (such as a command-line interface) or sometimes handling configuration via database settings.
* you can also edit configurations directly on the servers using a remote text editor such as Emacs or Vi.
* Handling configurations via databases is muchless flexible. application stack harder to get running. You need a working database to even start the application.
* Imperative command-line APIs is also a dubious practice for similar reasons but can sometimes be helpful,
* Puppet(step-by-step instructions for installing nginx, you declare the desired state), depend on being able to manage declarative configurations.
* if you are deploying to a production server, you want your connector to point to a production database server.
* some organizations try to handle this situation by managing their DNS servers, resolves to different servers depending on the environment.
* many ways for applications to invent their own configuration file formats, but there are a set

of base file formats that are popular. Such file formats include XML, YML, JSON, and INI.

Chatgpt simplified

Config base os

1. **Cross-Platform Functionality:**
   * Technologies like Java, Python, or Ruby offer cross-platform functionality, reducing visibility of operating system requirements.
2. **Low-Level Integrations:**
   * In industries like telecom, low-level hardware and software integrations make operating system requirements more apparent.
3. **Bare Metal Approach:**
   * Systems like Red Hat Satellite and Cobbler install the OS from scratch and set up base dependencies. Cobbler facilitates netbooting over the network.
4. **Base OS Images:**
   * Popular in cloud systems (e.g., AWS, Azure, OpenStack) and containers (e.g., Docker). Virtual machines or containers are created using reusable base images.
5. **Cloud and Container Systems:**
   * Cloud systems create virtual machines based on existing images, while containers declare base images and specify changes for customization.

Cluster

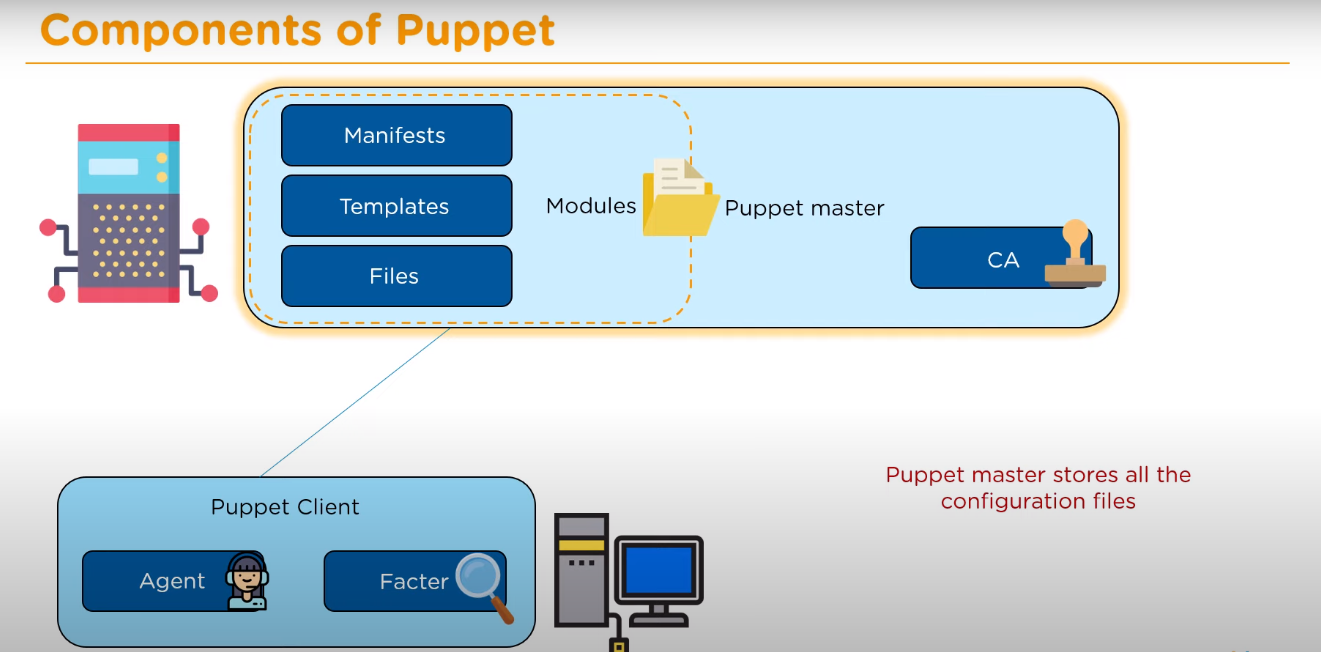
1. **Single Machine Scenario:**
   * In scenarios with a single machine and application, detailed cluster deployment may not be necessary.
2. **Reality of Distributed Applications:**
   * Organizations typically have applications distributed across multiple machines, whether physical or virtual.
3. **Configuration Management Tools:**
   * Tools like Puppet, Ansible, and Salt enable defining machine roles with associated packages and configurations.
4. **Container Systems:**
   * Docker offers infrastructure for describing connected containers and hosts. Cluster descriptors facilitate deployment.
5. **Cloud Systems:**
   * AWS and other cloud systems provide methods and descriptors for cluster deployments.
6. **Application Layer Description:**
   * Cluster descriptors are commonly used not just for infrastructure but also to describe the application layer.

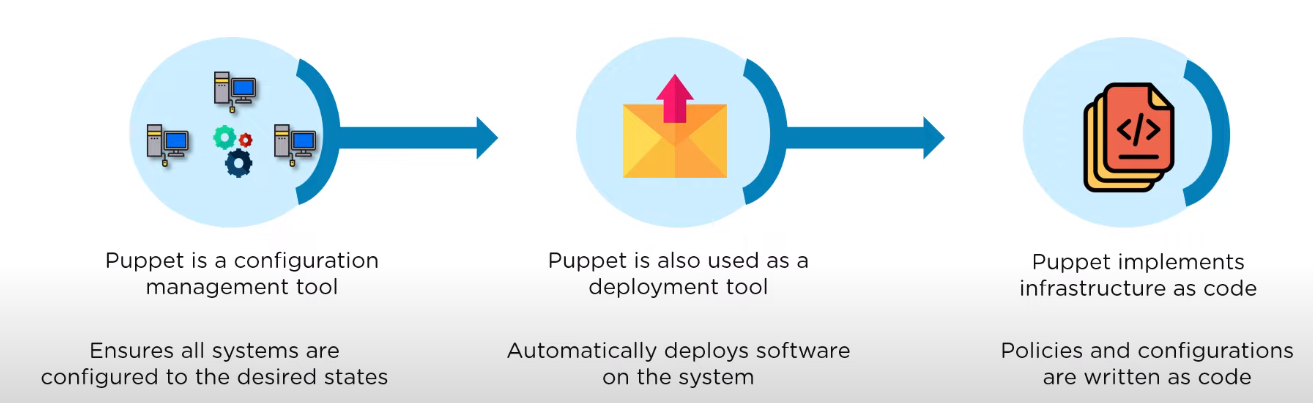
Delivering package to sys

1. **Package Installation:**
   * Configuration management systems often install application components as packages, maintaining system integrity through features like checksum verification.
2. **Package Delivery Methods:**
   * Delivery is commonly done through OS facilities (e.g., yum channels), and text-based configurations are a flexible approach.
3. **Text-Based Configurations:**
   * Managed in source code systems (e.g., Git) with tools like diff for debugging. Text-based configs are preferred over database-driven configurations.
4. **Database Configurations:**
   * Less flexible and considered an anti-pattern. Adds complexity to application stack and may require a working database for application startup.
5. **Command-Line APIs:**
   * Managing configuration via imperative APIs can hinder benefits of tools like Puppet, which rely on declarative configurations.
6. **Configuration File Formats:**
   * Popular formats include XML, YAML, JSON, and INI. Configuration files are often dynamic and require instantiation based on templates.
7. **Environment-Specific Configurations:**
   * Applications need adaptable configurations, like changing database server settings based on deployment environment (e.g., test vs. production).
8. **DNS Resolvers:**
   * Some organizations use DNS aliases for environment-specific configurations, allowing different server resolutions based on the environment.
9. **Developer Challenges:**
   * Developers may find it challenging to replicate DNS-based strategies locally. Providing configurable settings at the application level can be a simpler alternative.

Top of Form

puppet



Top of Form

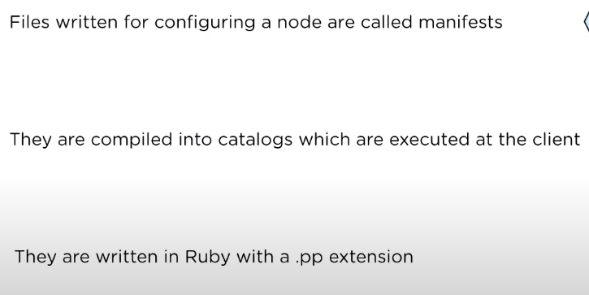
Working:

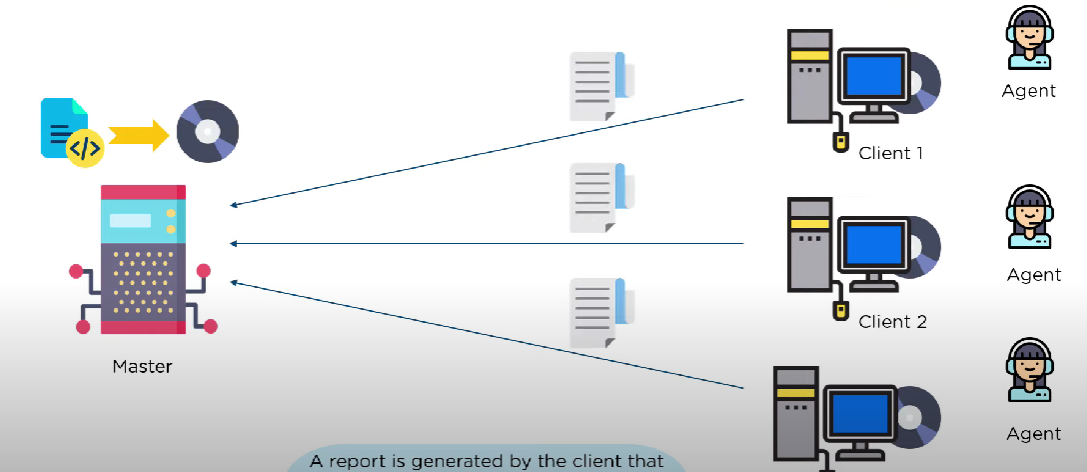
1)to est secure communication the client agents sends puppets the ca and p signs and sends them back and an agent constantly checks if theyre updated

2) after this the facter collects the state of the cliets and sends to master based on the facts sent the m compiles the manifests into catalogs

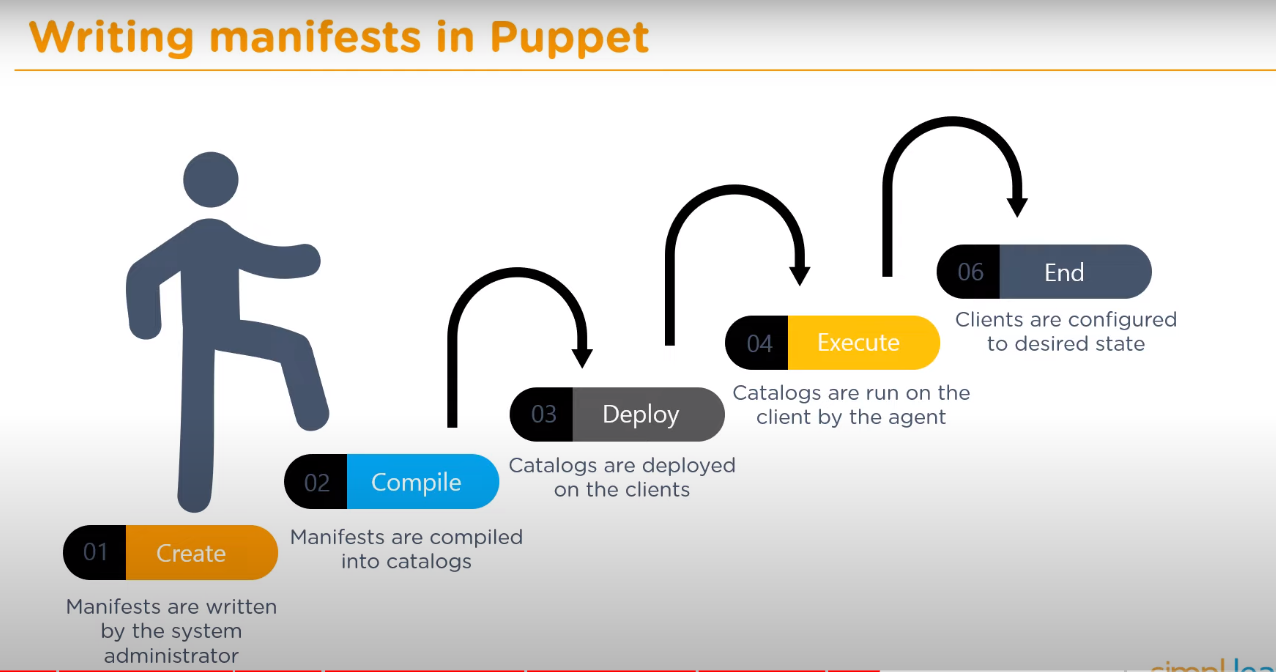
3)client will inistiate the catalog

4)a report is sent back to master that describes changes

Therefore master have full understanding of hw and sw



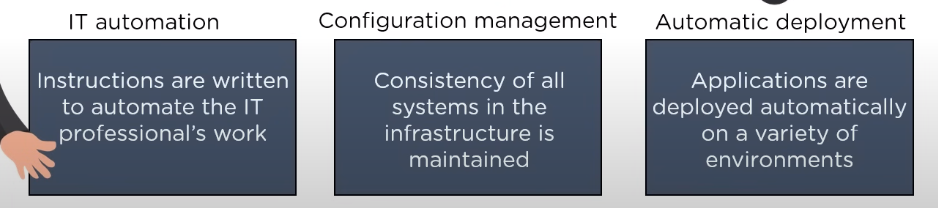
Spotify at&t google



Ansible

Manages infra of server on op side , if few v ez if more v hard

If server increase It becomes difficult to maintain consistency

Ansible helps in identical servers uses code to do that Use 1 script and have a consistent env

Create and control 3 areas:

Consit config

Automate deploy

It automate

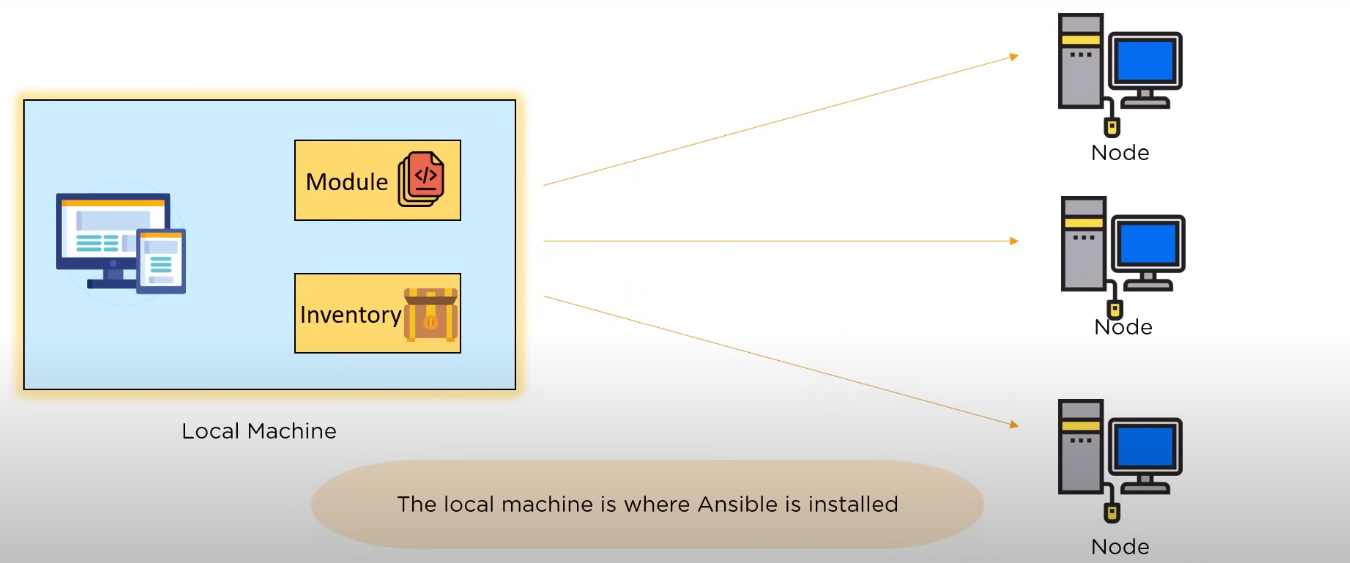
Has pull config

1 is key server client is saved

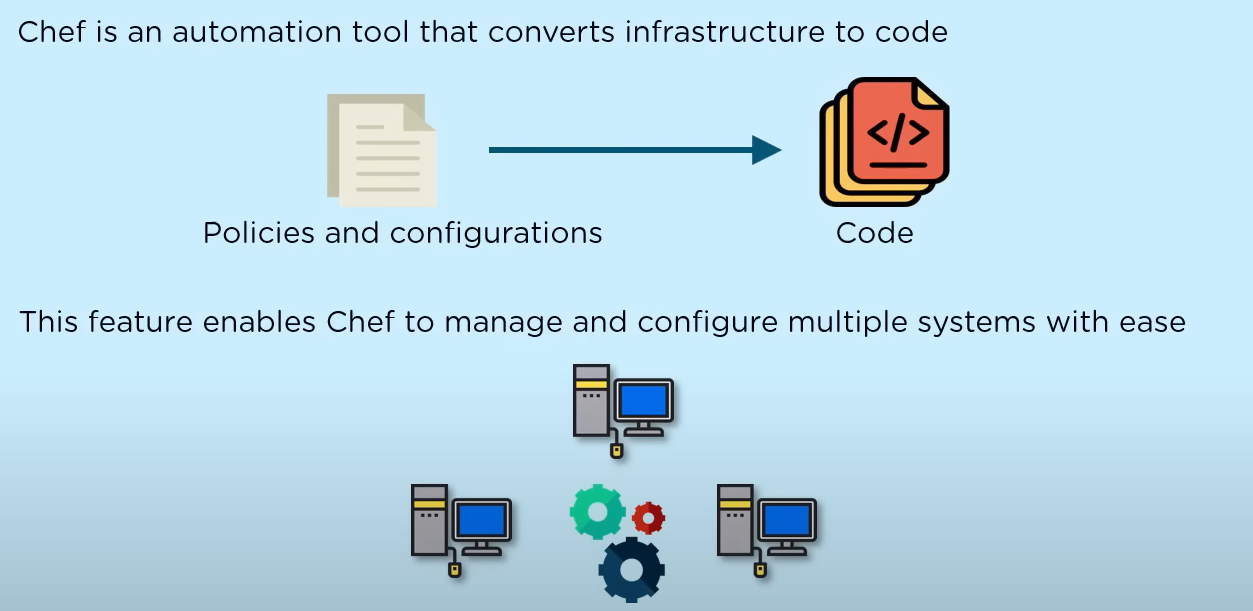
Has push config

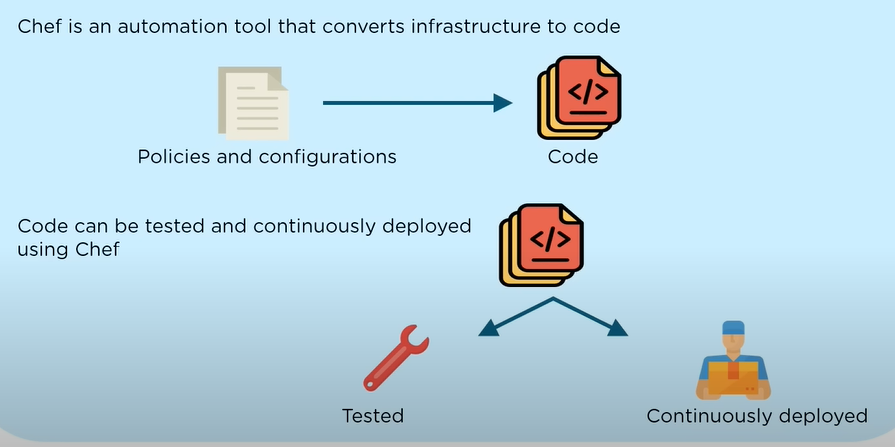
Forces to reconstruct structure on hw

Arch

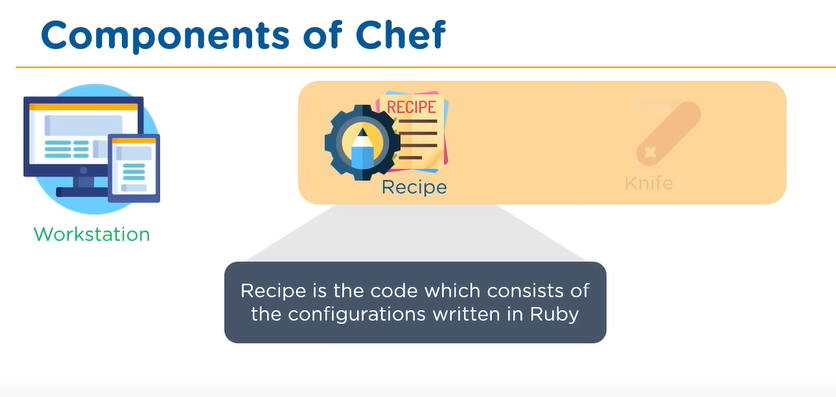


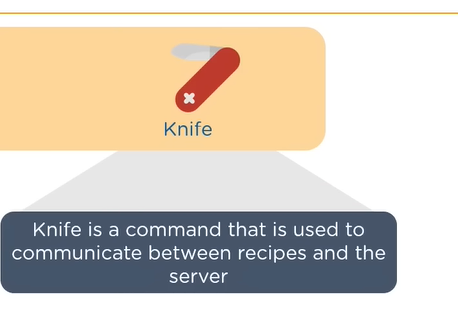
Chef:



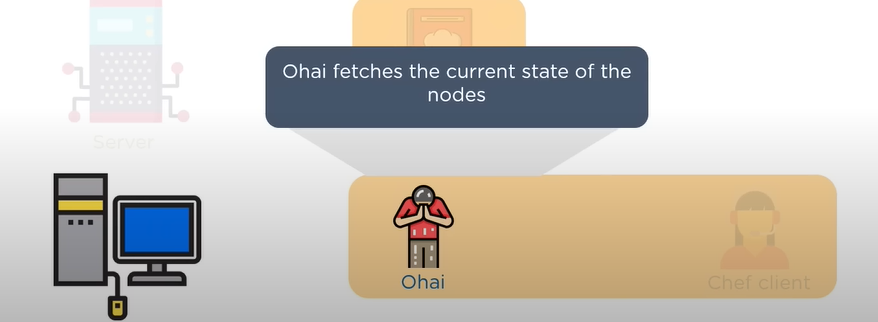


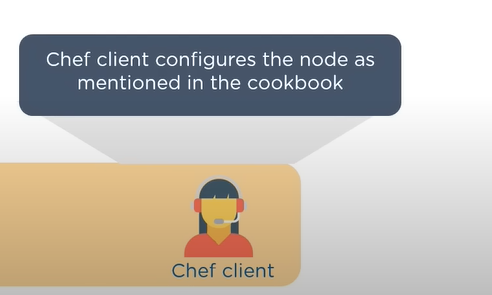
Components:



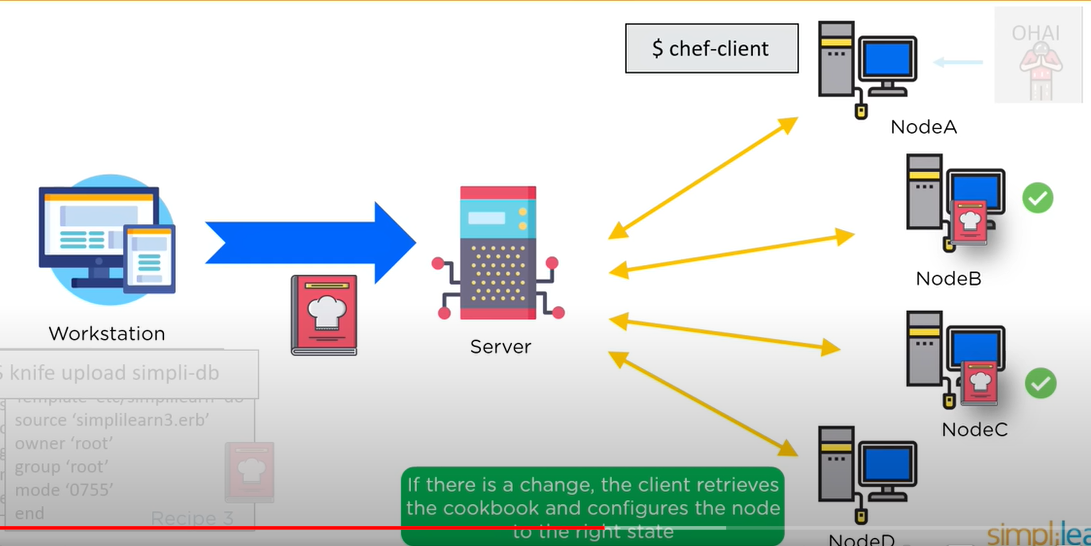






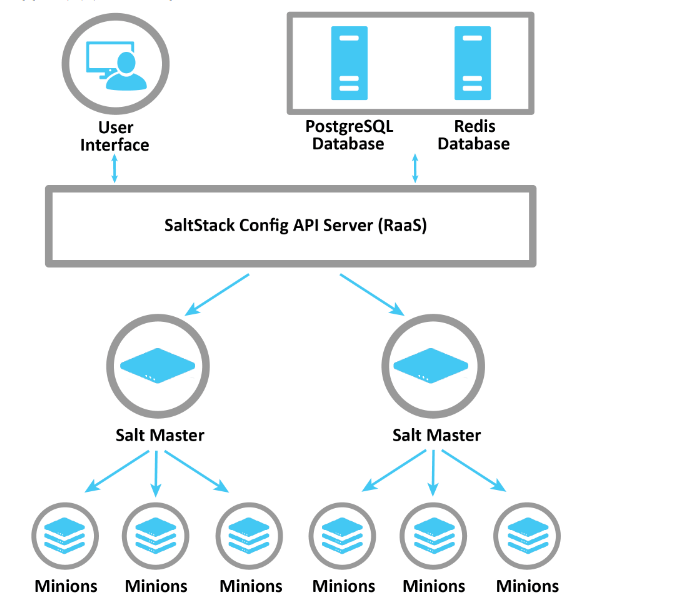


Arch:



Saltstack:

Arch:



Monitoring tools::

Munin

Ganglia

Nagios

Graphite

Log handling

Munin:

It is designed to be easy to use and set up. The out-of-the-box

experience gives you many graphs with little work.

Since Munin graphs statistics over time, you can see resource

allocation trends, which can help you find problems before

they get serious.

While Nagios focuses on the high-level traits of the health

of a service (whether the service or host is alive or not in

binary terms), Munin keeps track of statistics that it

periodically samples, and draws graphs of them.

• Munin can sample a lot of different types of statistics, from

CPU and memory load to the number of active users in your

children's Minecraft server. It is also easy to make plugins

that get you the statistics you want from your own services.

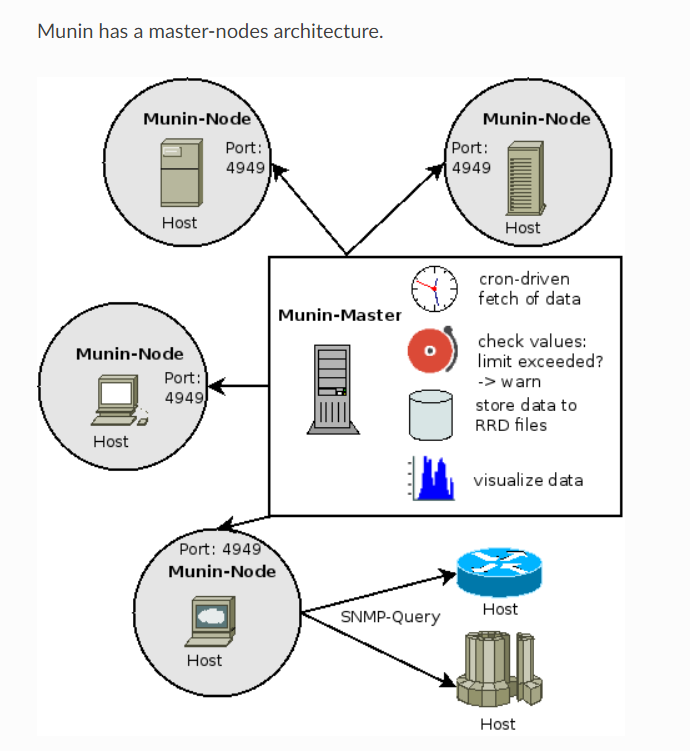
• An image can say a lot and convey lots of information at a

glance, so graphing the memory and processor load of your

servers can give you an early warning if something is about

to go wrong. 45

Arch:



There is a central server, the Munin master, which is

responsible for gathering data from the Munin nodes. The

Munin data is stored in a database system called RRD, which

is an acronym for Round-robin Database. The RRD also

does the graphing of the gathered data.

• The Munin node is a component that is installed on the servers

that will be monitored. The Munin master connects to all the

Munin nodes and runs plugins which return data to the master.