

Learn DevOps

Why DevOps

The screenshot shows a news article from Business Insider and a Google search result. The article title is "The Best Tech Skills To Have On Your Resume In 2015". Below it, a photo of Lisa Eadicicco, the author, and some social sharing options. The Google search bar shows "devops average salary". The search results page shows a snippet from Glassdoor stating the average DevOps Engineer salary is \$105,000.

BUSINESS
INSIDER

TECH

The Best Tech Skills To Have On Your Resume In 2015

Lisa Eadicicco
Jan. 22, 2015, 12:16 PM 18,969 6

Google devops average salary

Web Images News Videos Maps More Search tools

About 99.700 results (0,53 seconds)

\$105,000
The average Devops Engineer salary is \$105,000 .
Salary: Devops Engineer | Glassdoor
www.glassdoor.com/Salaries/devops-engineer-salary-SRCH_K00,15.htm

- DevOps is one of the top skills employers are looking for in the IT industry (BusinessInsider)
- The yearly average DevOps Engineer salary is \$105,000 (Glassdoor)
- In San Francisco the average is even higher, about \$146,000 (Indeed)
- Some even earn \$250,000+ (Quora: Salary of DevOps Engineer)

Why DevOps

BUSINESS
INSIDER

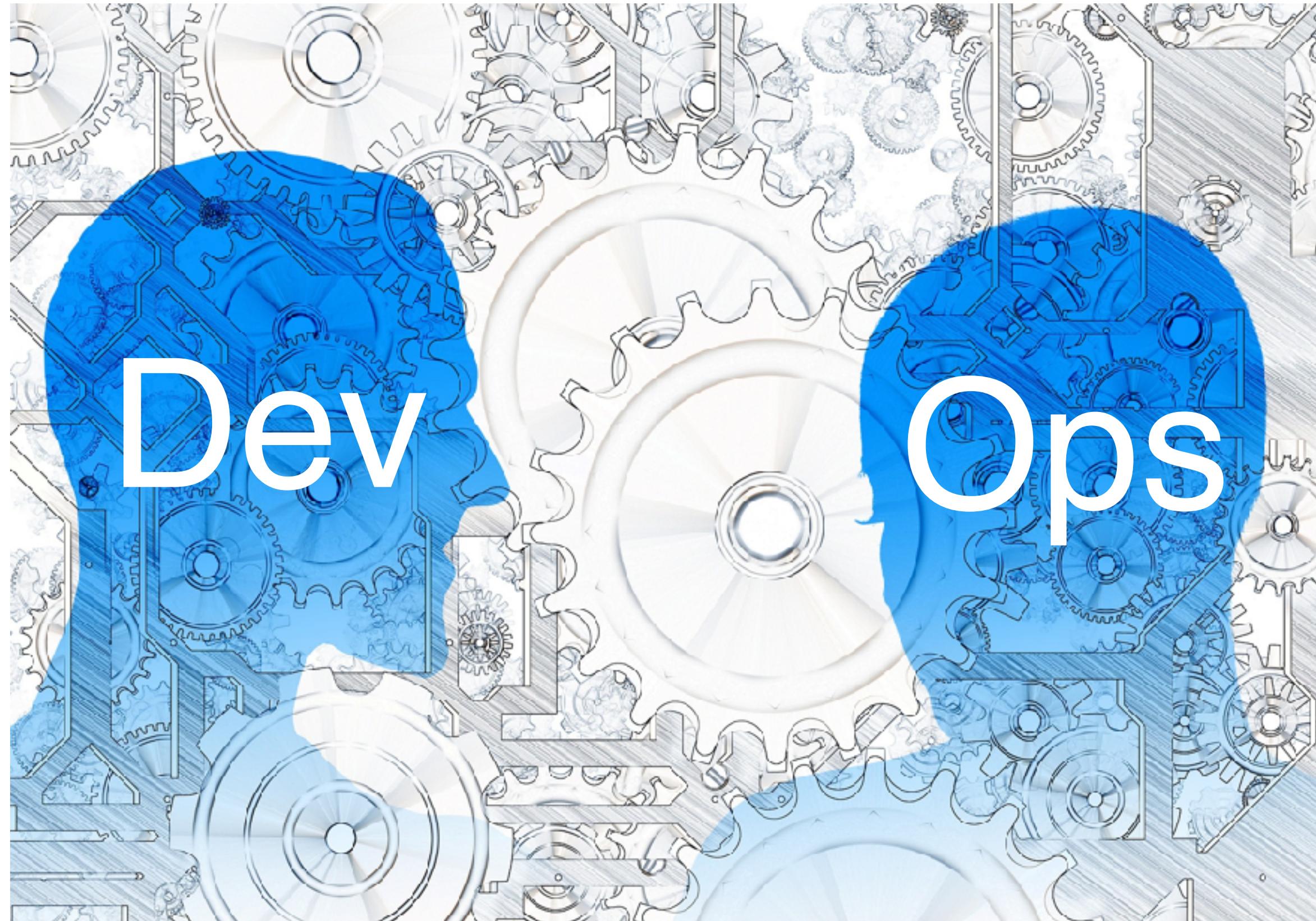
ENTERPRISE

Companies are racing to change how they build software – and a company called Chef is at the heart of it

 Matt Weinberger  
Apr. 1, 2015, 3:00 PM  4,526  1

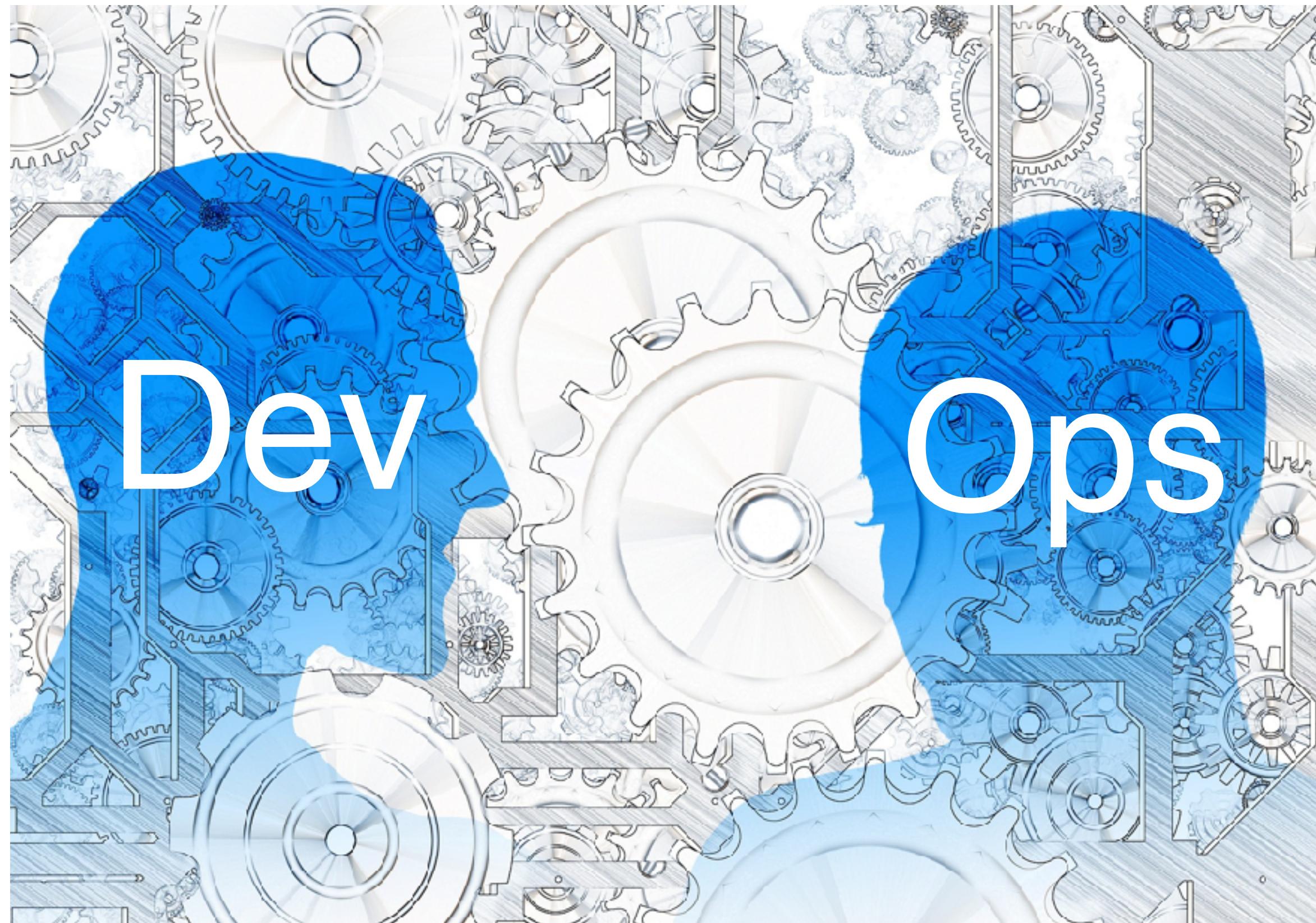
- All big tech companies are making software better constantly and continuously, at an enormous phase.
- Chef has raised \$65 million in venture capital funding since it was founded
- Puppet Labs has raised \$85.5 million
- Ansible has been acquired by Red Hat in October 2015

Learn DevOps



- I will teach you how to excel in software delivery and apply the same methodologies that made the big Tech companies successful
- This course will help you to boost your career
- Having these skill on your resume will potentially increase your salary
- You will be able to make your company more efficient, and be able to continuously deliver better software

Learn DevOps



- I will show you how the technologies work together, this course is not just a one-by-one overview of DevOps tools
- I put a lot of effort in this course to provide you with not only the theory, but also demos you can try out on your own machine
- I will provide support for those who can't complete a lab or have questions. We have discussion forums and a Facebook group

Course Outline

Concepts	Step 1: Provisioning	Step 2: automation and Configuration Management	Step 3: Continuous Integration	Step 4: Deployment	Step 5: Continuous Monitoring	Microservices & Containerization
What is DevOps	Vagrant	Ansible	Continuous Integration	Artifactory	Nagios	The Twelve factor app
Version Control		Chef	Jenkins	Ubuntu repository	Application Monitoring	Microservices
Continuous Delivery		AWS OpsWorks	build, test, package	chef deployment	ELK stack	Containerization
Continuous Deployment						Container Orchestration
Configuration Management						Kubernetes
Provisioning & Cloud						

Introduction

Course Outline

Concepts	Step 1: Provisioning	Step 2: automation and Configuration Management	Step 3: Continuous Integration	Step 4: Deployment	Step 5: Continuous Monitoring	Microservices & Containerization
What is DevOps	Vagrant	Ansible	Continuous Integration	Artifactory	Nagios	The Twelve factor app
Version Control		Chef	Jenkins	Ubuntu repository	Application Monitoring	Microservices
Continuous Delivery		AWS OpsWorks	build, test, package	chef deployment	ELK stack	Containerization
Continuous Deployment						Container Orchestration
Configuration Management						Kubernetes
Provisioning & Cloud						

Course objective

- To excel in software delivery and apply the same methodologies that made the big Tech companies successful
- To make you familiar using technologies
 - Chef
 - Ansible
 - Jenkins
 - Docker, Kubernetes

Practice

- A lot of slides provide practical information how to do things
- Use the git repositories and procedure documents, provided in the text documents in this course
- You can copy paste commands from those documents and use the git repositories to try out the demos yourself
- I will keep the repositories up to date with new and extra information

Feedback and support

- To provide feedback or get support, use the discussion groups
- We also have a Facebook group called Learn DevOps: Continuously Deliver Better Software
- You can scan the following barcode or use the link in the next document after this introduction movie



DevOps Concepts

What is DevOps?

What is DevOps

Dev: “It works on my machine”

Dev: “worked fine in DEV, OPS problem now”

What is DevOps

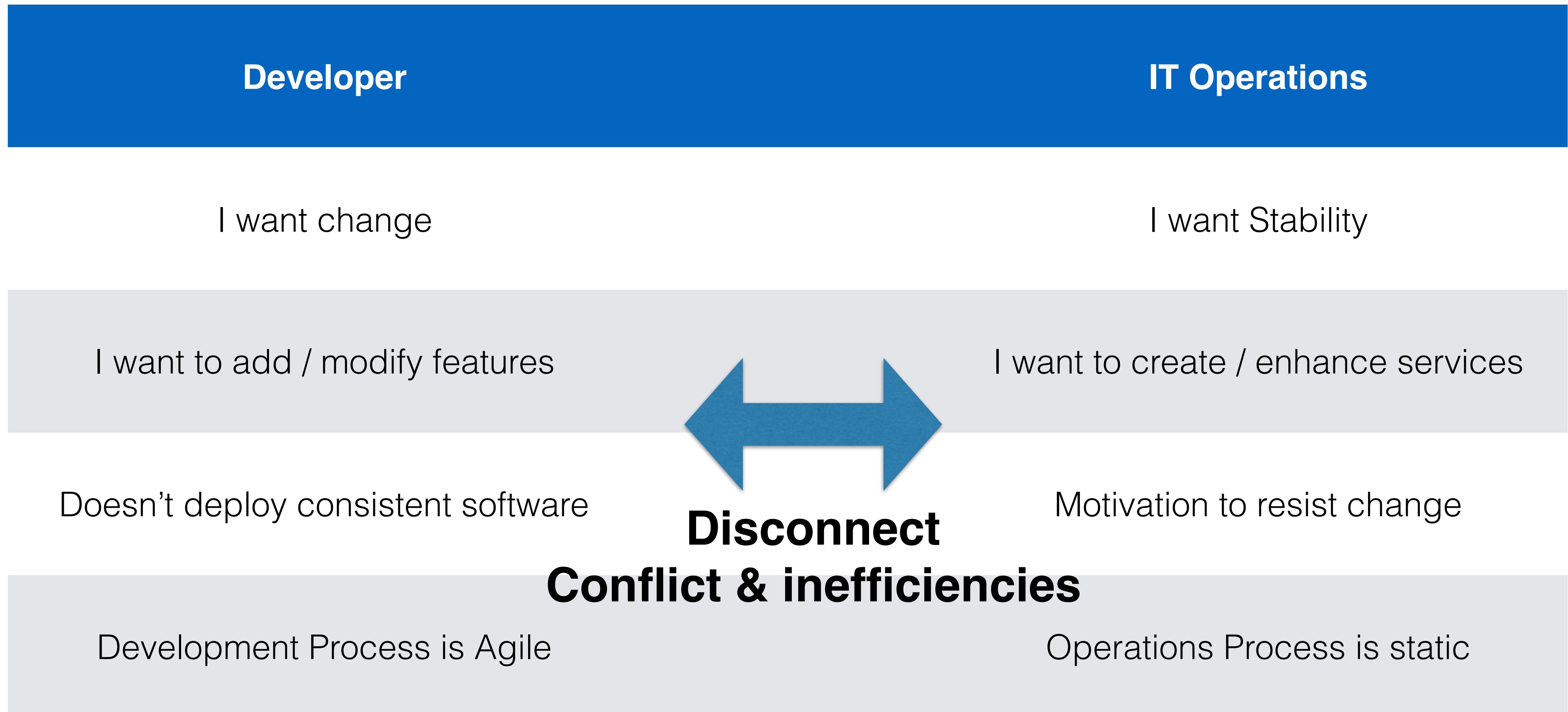
Ops: “push code and pray”

Ops: “It’s not the server, it’s the code”

What is DevOps

“us and them”
(developers and Sysadmins)

Developer vs. Operations

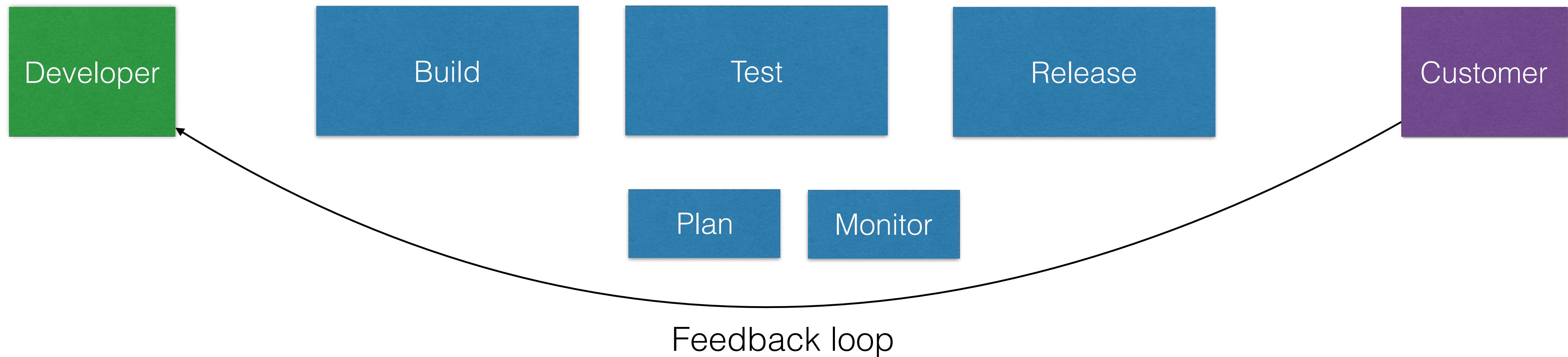


What is DevOps?

- DevOps is an approach to bridge the gap between software development and operations
 - It is a response to the interdependence of software development and IT operations
 - DevOps is not a technology problem, but a business and cultural problem
 - Stresses collaboration between development and operations
 - Aims to deliver software and services quicker

Continuous Delivery

DevOps can only work when the Software Development Life Cycle (SDLC) can support it



- DevOps: Complete the full SDLC as quickly as possible
- Works only well if development team and Operations work well together

DevOps is a transformation

- Teams need to get educated
 - ➔ Buy-in
 - ➔ Change Management Process
- DevOps is what Agile was earlier
 - ➔ The next level
- High Performing IT organizations are more agile when using DevOps
 - ➔ Higher deploy frequency
 - ➔ Lower Mean Time to Recovery
 - ➔ Lower Lead time for Changes
 - ➔ More reliable software
 - ➔ Example: Amazon was able to do 50 million deployments in one year (AWS reInvent 2015)

Benefits to the business

- Shorter time-to-market
- Releases are quicker, more stable
- Better quality
- Better collaboration between development and operations
- Faster learning whether a new feature is wanted by the customer
- Less white board planning with no data

Benefits to the customer

- New features delivered to the customer quicker
- A leaner organization
- Batches become smaller, agile is starting to make sense for the customer
- Better visibility of new features

Lean Thinking

- The Lean Organization:
 - Get ideas in production fast
 - Have (some) customers using new features
 - Get validated feedback
- Lean Thinking needs DevOps to get out software quickly

A change of culture

DevOps is more than just tools: it's a practice of Dev and Ops engineers participating together in the entire service lifecycle (SDLC)

Tools can help

- Cloud computing (AWS, Azure, Google Cloud)
- Infrastructure as code (Chef, Puppet, Ansible, SaltStack)
- Build & Test using Continuous Integration (Jenkins)
- Containerization (Docker)

DevOps as a Service

- Development Environment as a Service in the cloud (c9.io)
- Build tools as a Service (Atlassian)
- Provisioning and deployment tools as a Service (AWS OpsWorks)
- Platform as a Service (Heroku)

Version Control

Version Control

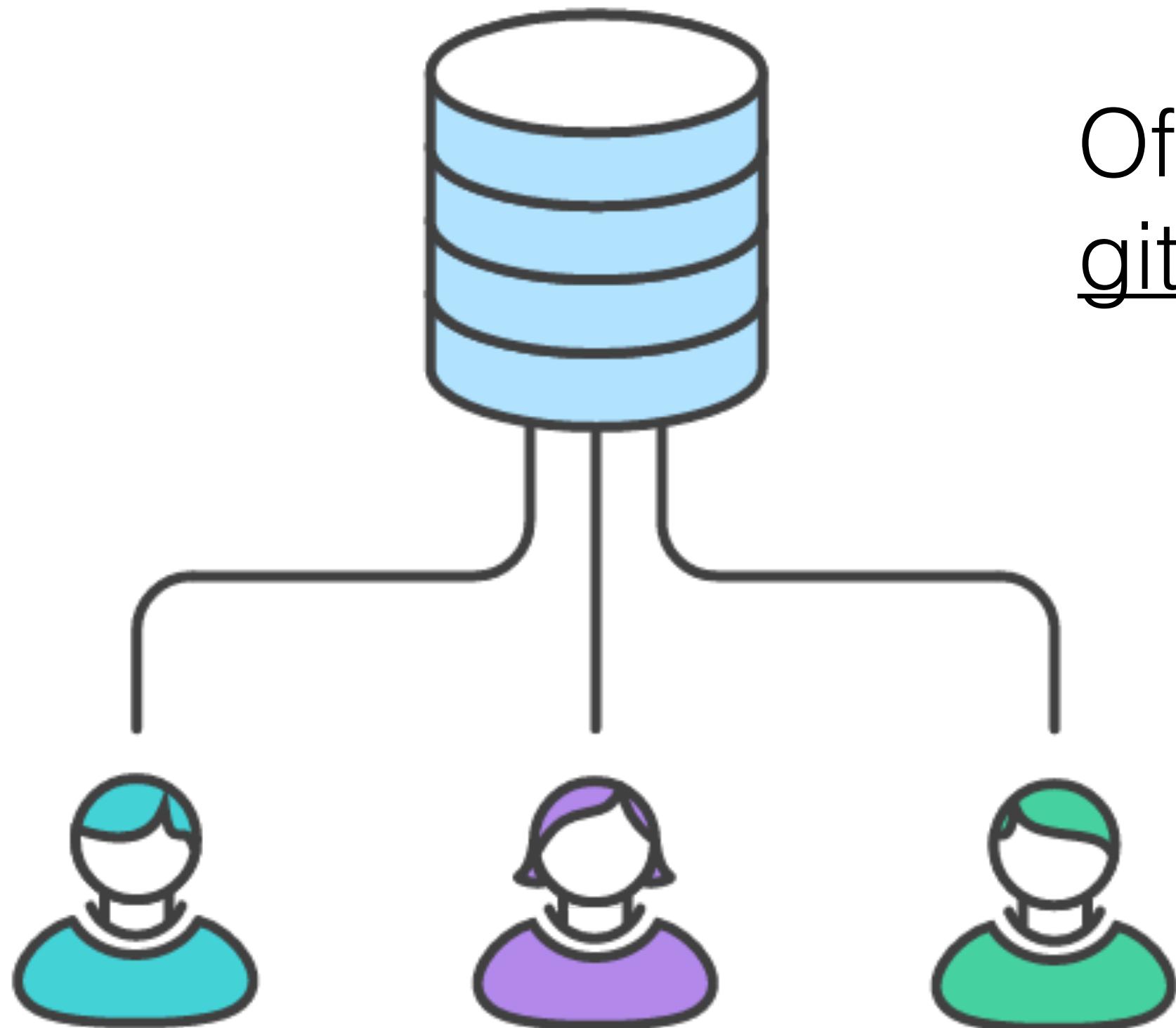
- Version control, also known as revision control or source control, is the management of changes to documents, computer programs, large web sites, and other collections of information
- Previously: CVS or Subversion (SVN)
- Now: GIT
 - Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.
Initially designed and developed by Linus Torvalds for the Linux Kernel

Why version control

- Will be used by Devs as golden source for the software product
- Will be used by Ops as version control for automation
 - Scripts
 - Tools
 - Software Configuration

GIT

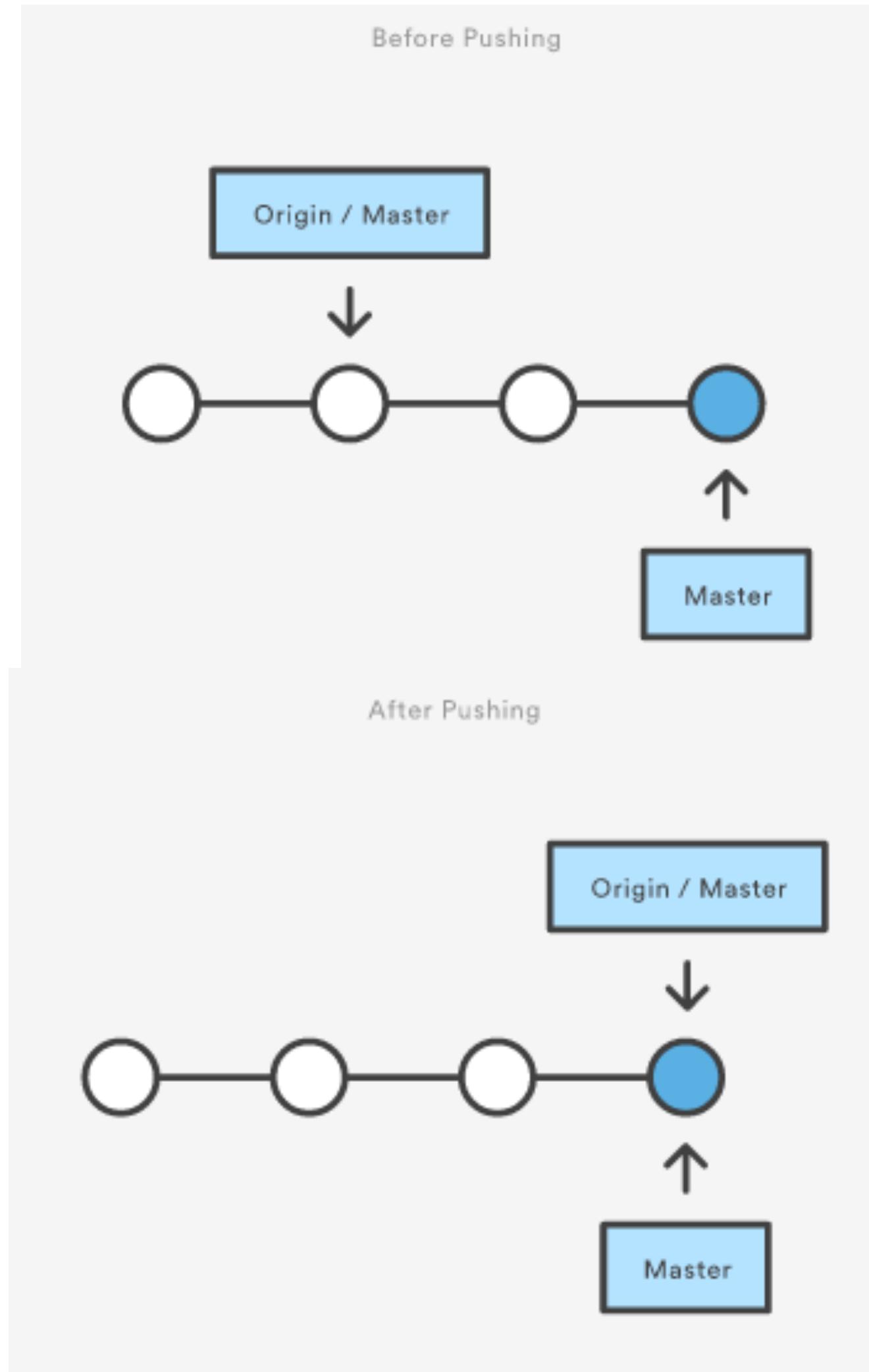
- A distributed version control



Often a central repository like
github.com or bitbucket.org

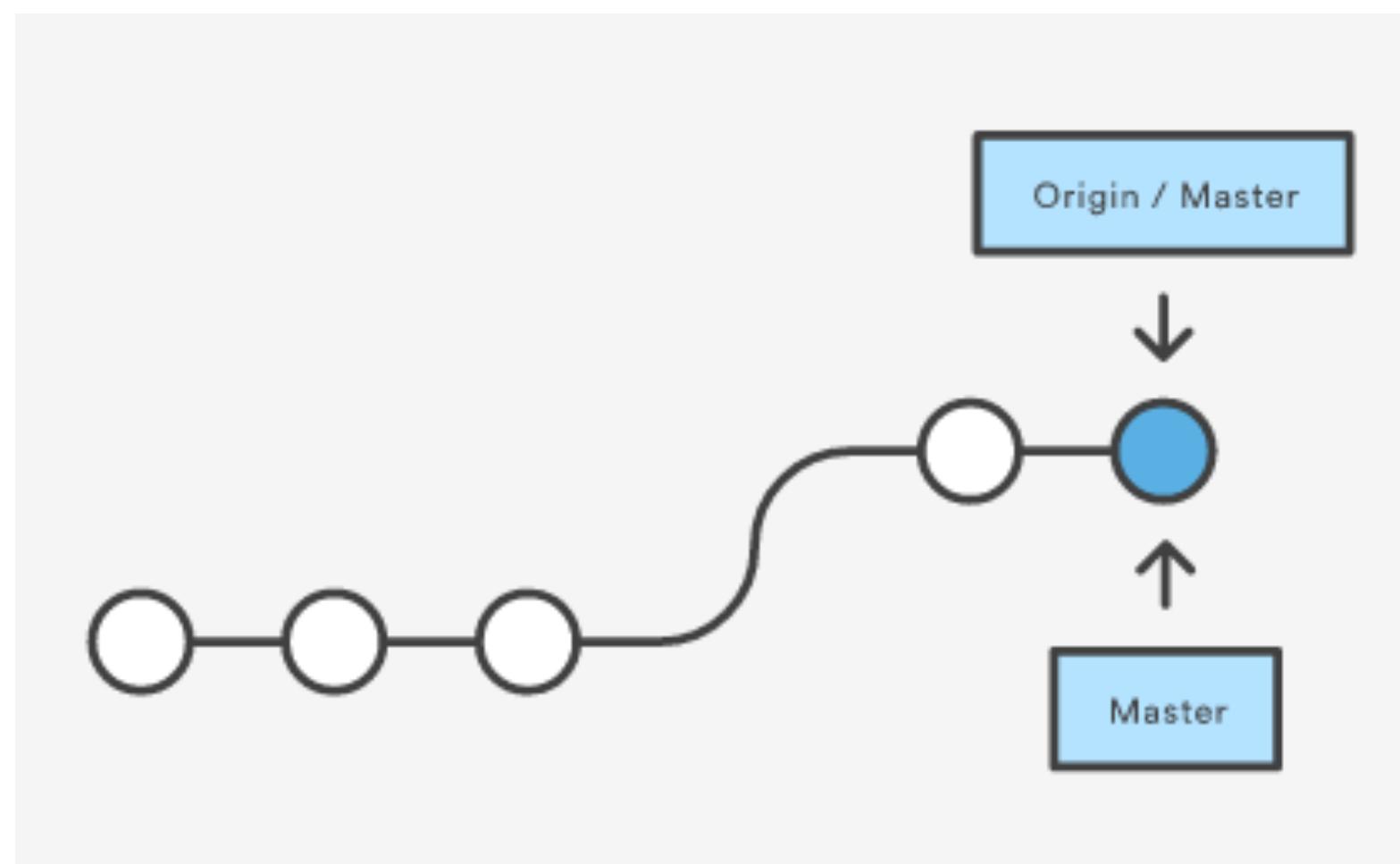
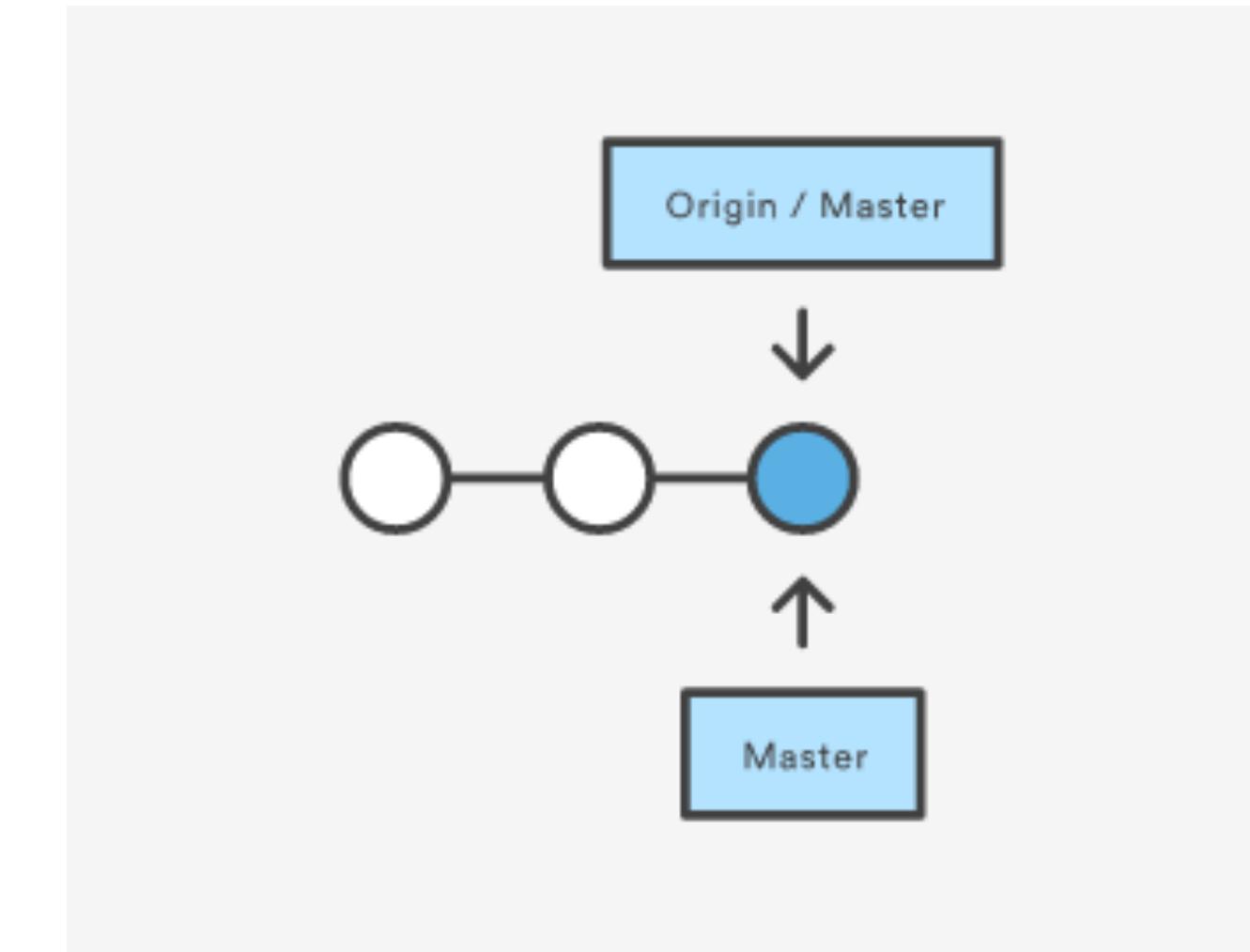
Developers have a full local
copy of the repository, they
“clone” the central repository

Making changes



- As a developer, you can make some local changes
- You then commit those changes locally using “git commit”
- To share the changes with your team, you use “git push” to send your changes to the central repository

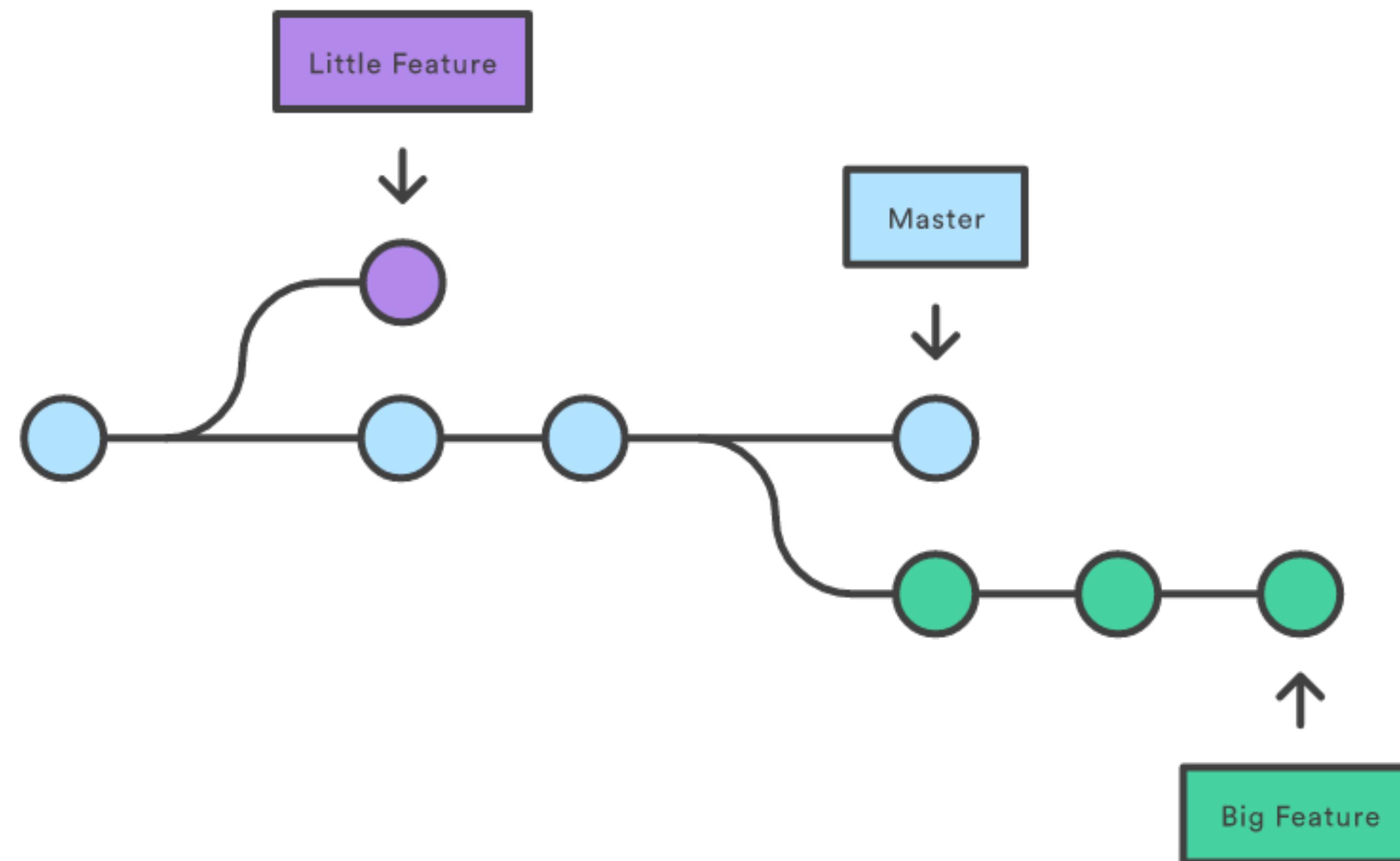
Retrieving changes



- Another developer can now retrieve the changes by using “git pull”
- This will fetch the changes from the central repository and merge the remote changes with your local branch

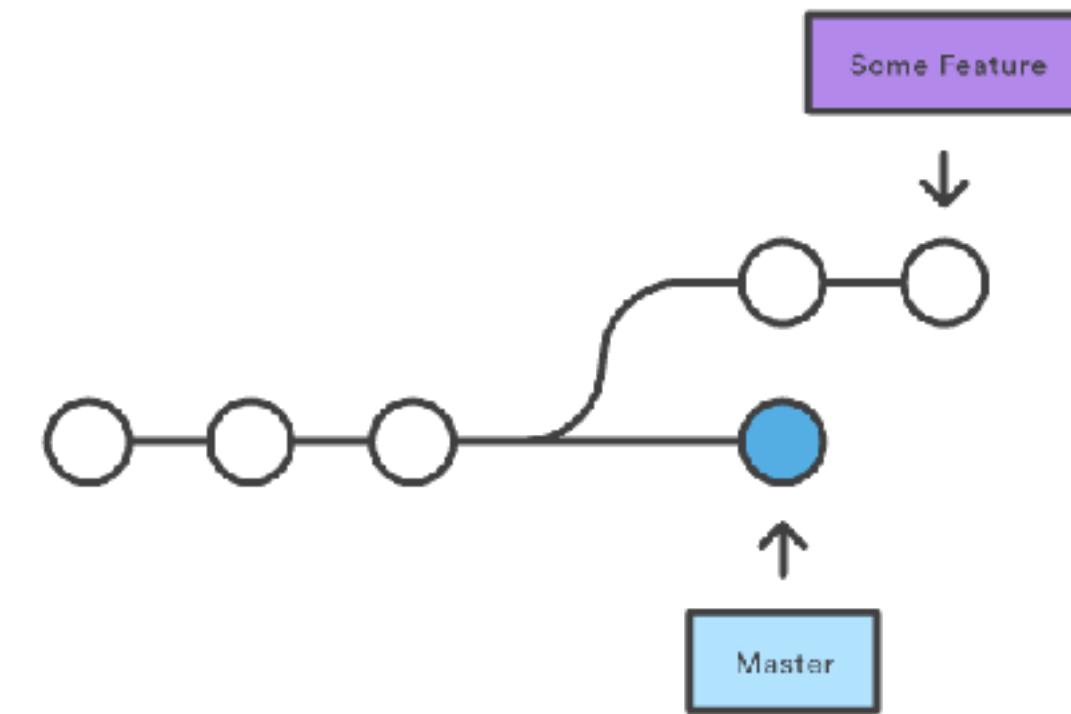
Branches

Developers can work on their own feature branches



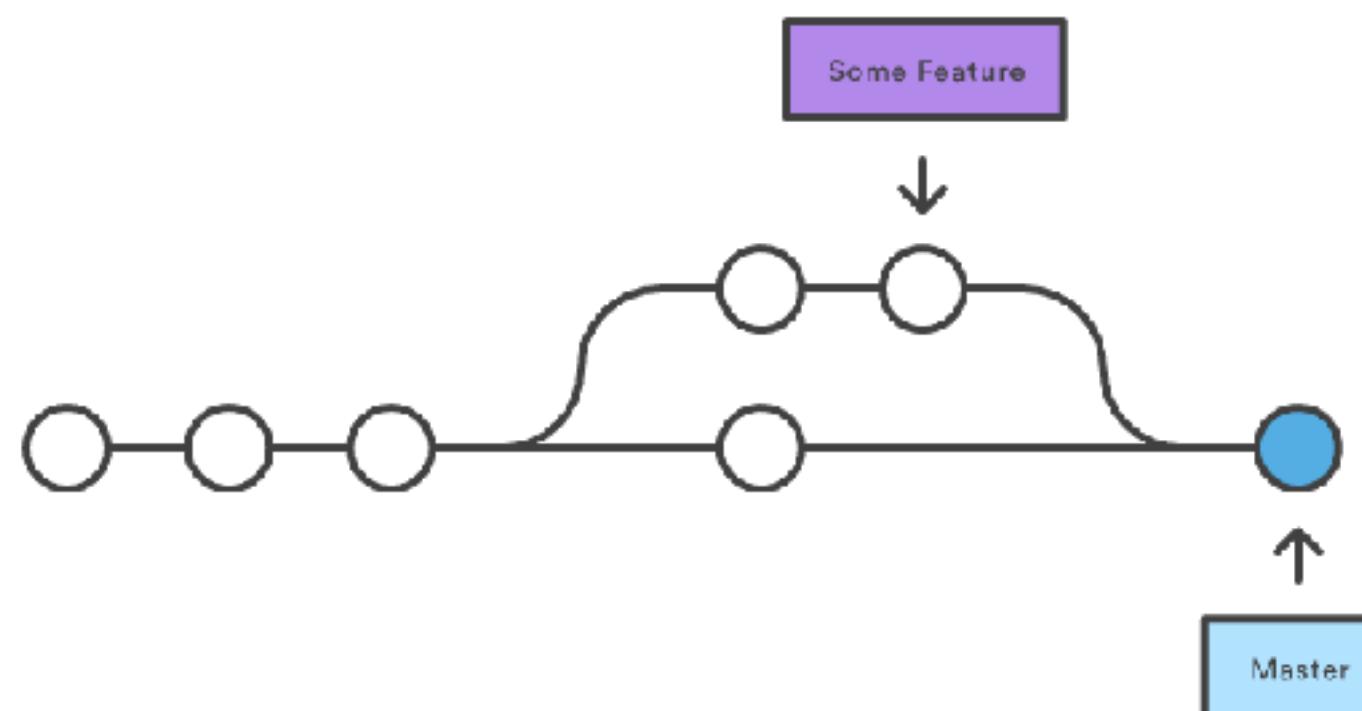
Merging

Before Merging



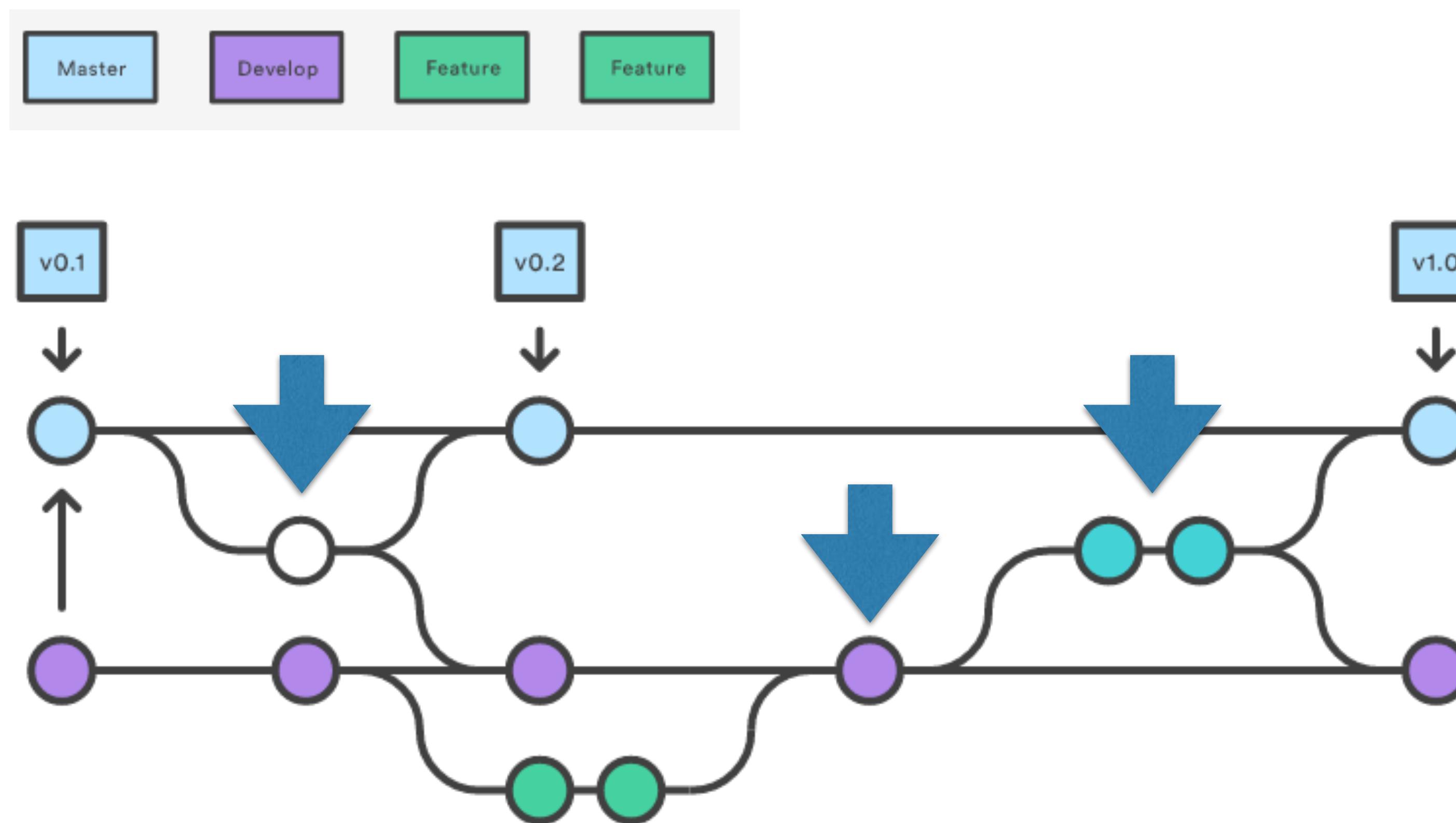
- After a feature is finished the branch can be merged
- This can be done using the command “git merge”

After a 3-way Merge



Git Flow

- The Gitflow Workflow defines a strict branching model designed around the project release



GIT Tools

- Commandline: git (Linux / Windows / Mac)
- SourceTree (Windows / Mac)
- GitHub (Windows / Mac)
- See also <http://git-scm.com/downloads/guis>

DEMO

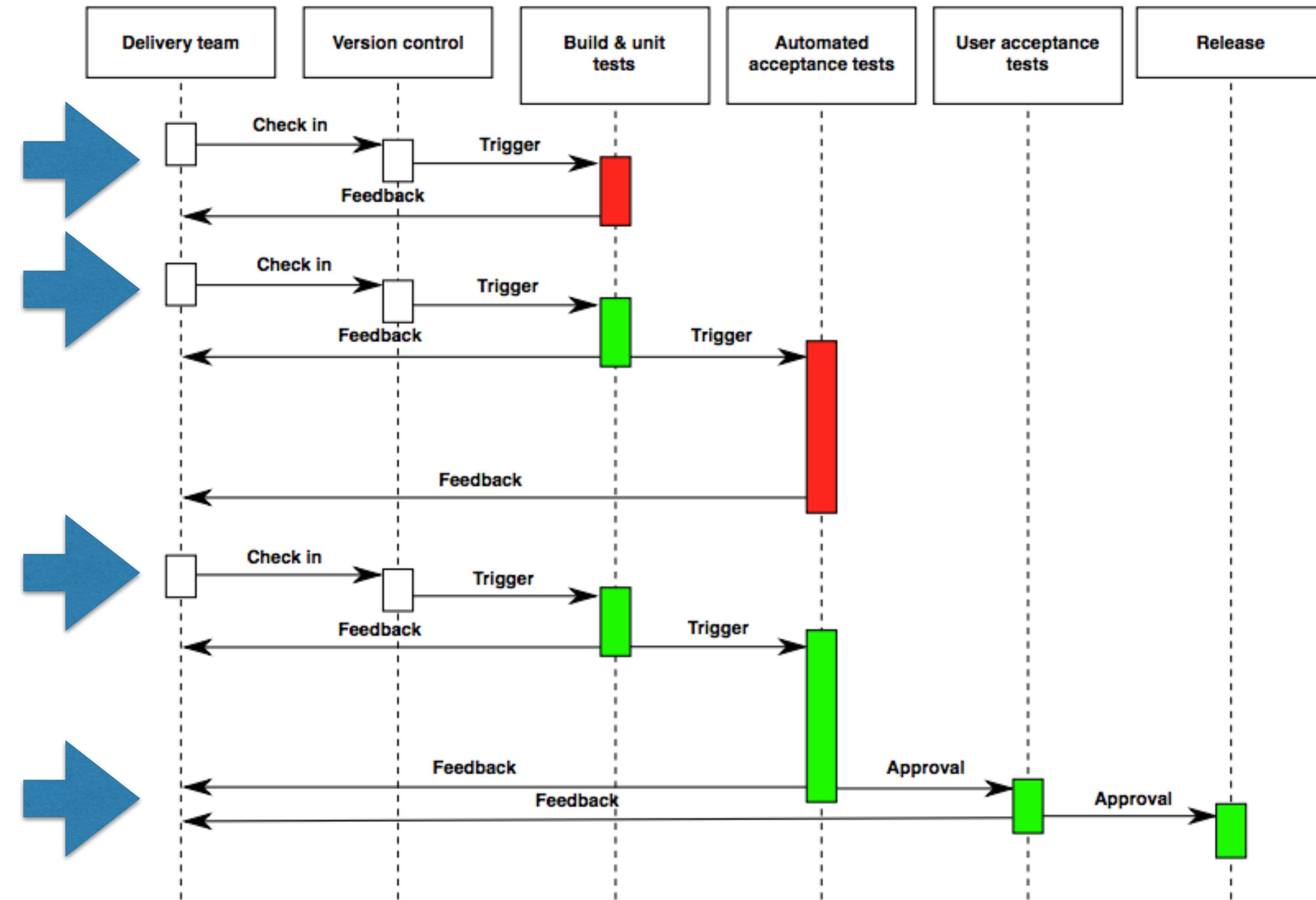
Using GIT

Continuous Delivery

What is CD

- Continuous Delivery (CD) is a set of practices and principles in software engineering aimed at, building, testing, and releasing software, faster and more frequently. These principles help reduce the cost, time and risk of delivering changes, and ultimately value, to customers by allowing for more incremental changes to applications in production (wikipedia)

The CD Process



Benefits

- Accelerated time to market
- Building the right product
- Improve productivity and efficiency
- Reliable releases
- Improved product quality
- Improved Customer Satisfaction

Continuous Integration

- Continuous integration (CI) is the practice, in software engineering, of merging all developer working copies to a shared mainline several times a day. (Wikipedia)
- Subset of Continuous Delivery
- In Practice: trigger automated builds & tests at least once a day for the developers' commits
- Provides feedback loop back to the developer to fix build errors

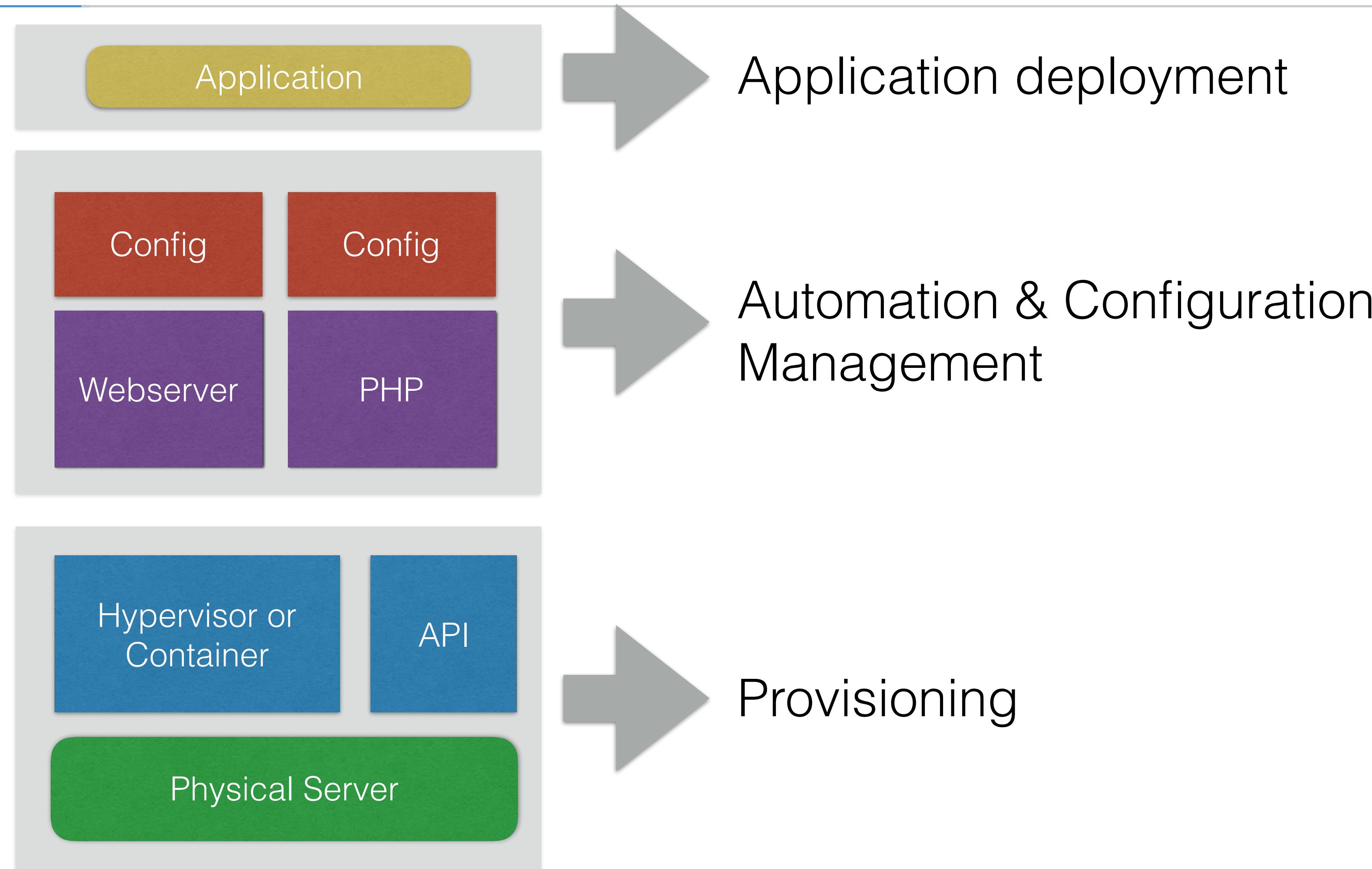
Continuous Deployments

Continuous Deployments

- Continuous Deployments is the automated deployment or release of production ready code
- The next step of continuous delivery
- Automation of the deployment process
 - Deployment should only take minutes
 - Preferably by a click on a button
- New hires at Facebook push code in production on their first day

Configuration Management and Automation

Provisioning vs. Configuration Management



Configuration Management

- Configuration management (CM) is a systems engineering process for establishing and maintaining consistency of a product's performance, functional and physical attributes with its requirements, design and operational information throughout its life. (Wikipedia)
- In our context: maintaining consistency of our software configuration

Why Automation

- Infrastructure becomes unmanageable
- Too much change
- Outdated documentation
- No way to easily recreate the environment
- The answer: Automation and Configuration Management
 - Tools: Chef, Puppet, Ansible, SaltStack,

Benefits

- Automation tools will help us deliver and maintain software on our servers, as code
- Infrastructure becomes code
 - ➔ It becomes maintainable
 - ➔ Version controlled
 - ➔ Testable
 - ➔ Collaborative

Guidelines

- If possible, try to use Infrastructure as a Service (IaaS)
 - Programmable Provisioning
 - More Efficient
 - Better Manageable
 - Simpler

Guidelines (cont.)

- Be a Developer: use version control (GIT)
- You can't automate what you don't understand
- Make use of automated testing before deploying

Provisioning

Provisioning

- Huge difference between on-premise and cloud
- Cloud is in general more flexible
 - On-demand resources
 - Great APIs
- You might not have a choice
- Hybrid solution (partially cloud, partially on-premise)
 - Example: Dev & Staging on cloud, prod on-premise

On-premise provisioning

- Creation of VMware / Xen / KVM / HyperV instances
- Can be managed by software like OpenStack:
 - OpenStack software controls large pools of compute, storage, and networking resources throughout a datacenter, managed through a dashboard or via the OpenStack API.

Cloud provisioning

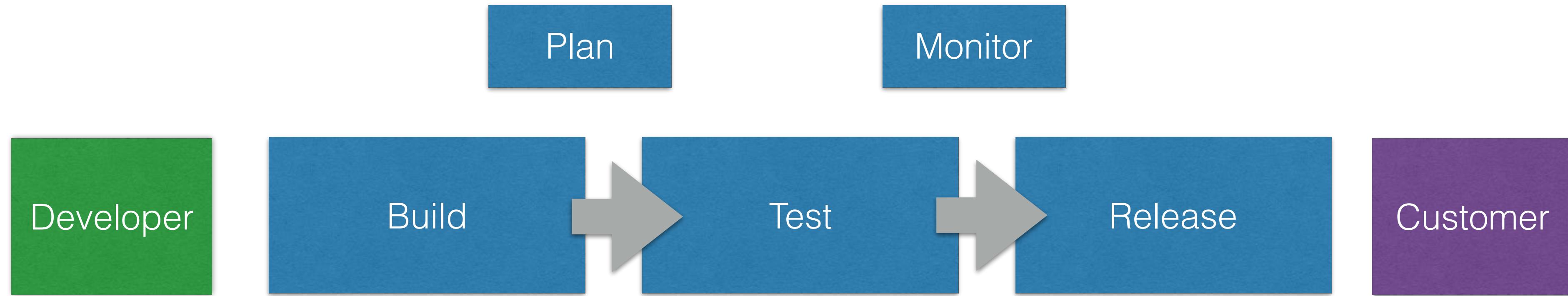
- Cloud providers expose APIs
 - Amazon AWS, Azure, Google Cloud, Digital Ocean
- Instance provisioning is done in minutes
- Resources are on-demand
- Instances can be started and stopped easily, with pay-as-you go pricing
- Scale based on time and load

Common Service types

- IaaS - Infrastructure as a Service
 - Cloud provider will provision a Linux / Windows server, you need to maintain OS and configuration yourself
- DBaaS - Database as a Service
 - Managed database, you don't need to maintain underlying OS
- PaaS - Platform as a Service
 - OS is managed for you. You deploy software package

Plan & Monitor

Plan & Monitor



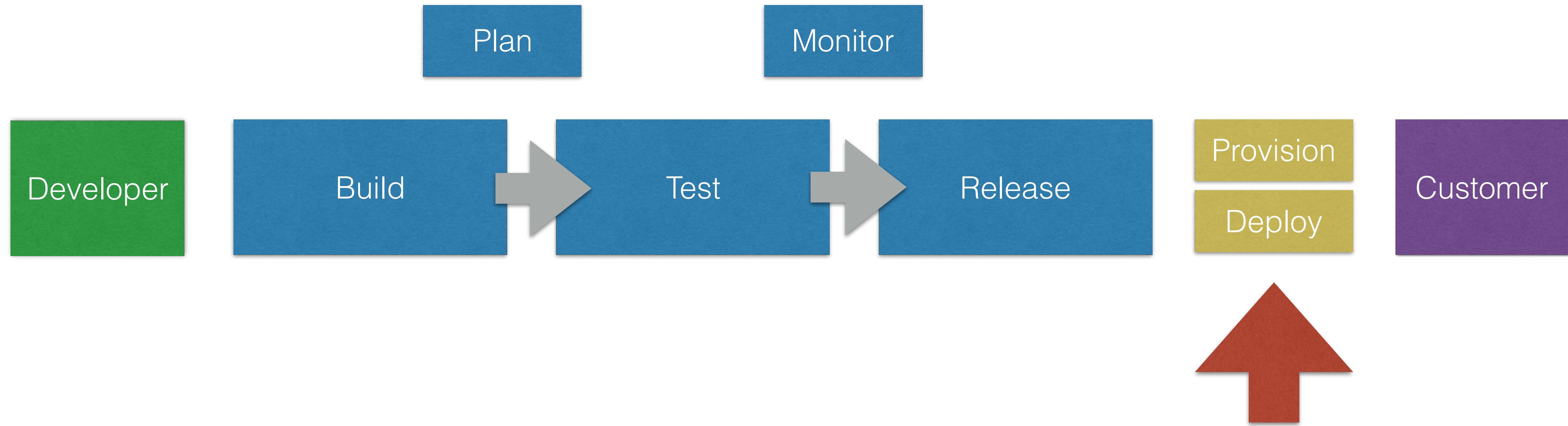
- The Operations team should plan & monitor progress using the same methodologies development teams
- Operations should follow an agile approach
 - Scrum
 - Kanban

Plan & Monitor

- In practice scrum is often used in larger organizations
 - All tasks should be logged in a central system (e.g. JIRA)
 - Every day stand-up meetings to monitor progress, identify blockers
 - Tasks should ideally be broken up to 1-day tasks
 - e.g. develop recipe for app deployment
 - Pre-planned sprint runs over the course of a couple of weeks
 - end-of-sprint demos shows work done by team during sprint
 - See Agile / Scrum topics or books

Step 1 - Provisioning

SDLC

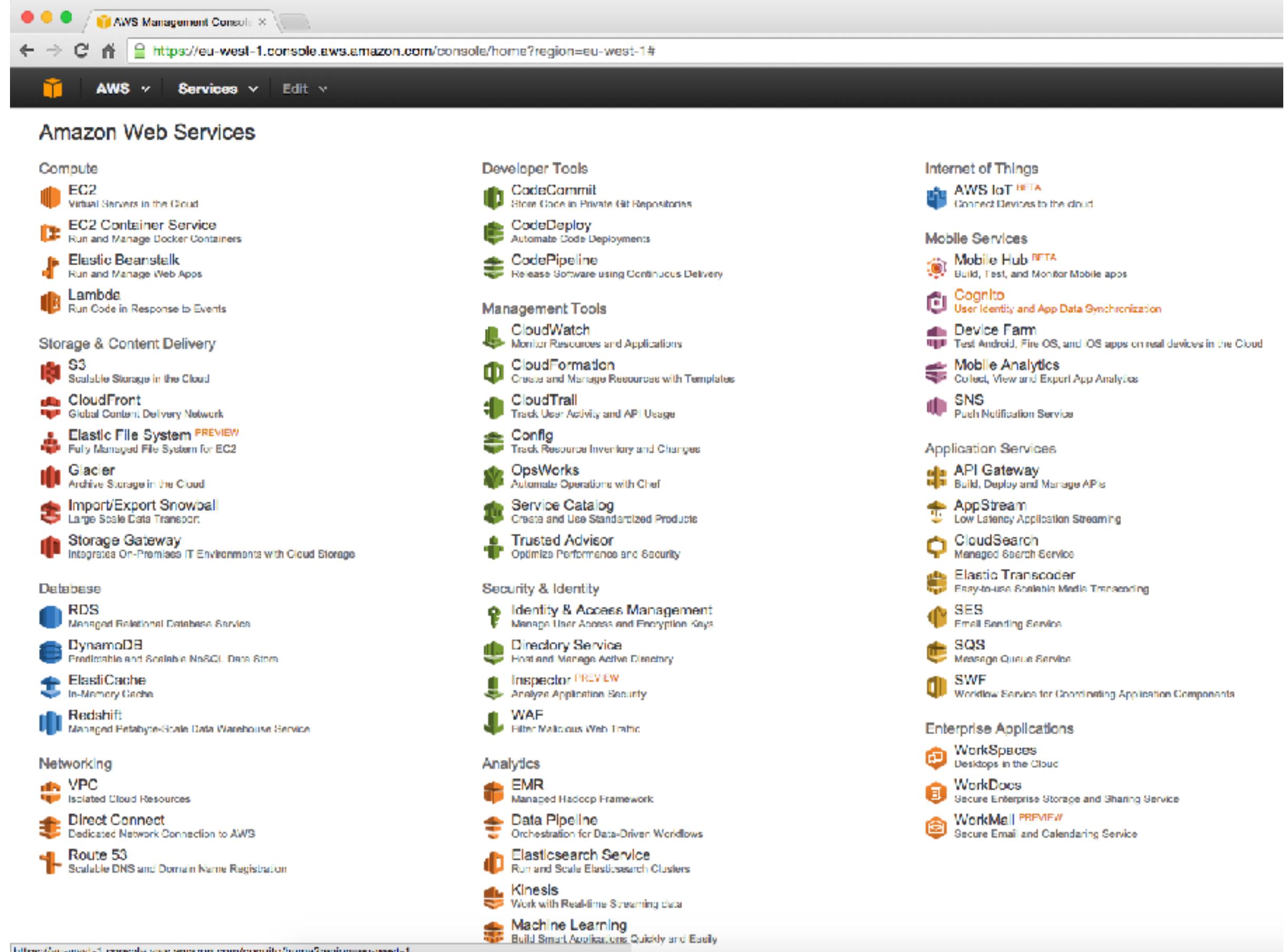


- Provisioning is creating the machines with OS and software
- Deployment is installing and configuring your software product on the provisioned machine

On-premise provisioning

- Vagrant: quickly provision machines on your own machine
 - Wrapper around VMWare, VirtualBox, KVM, HyperV, and more
- Enterprises: OpenStack or proprietary tools, often VMWare based
 - Unique procedures to provision dev / staging / prod machines

Cloud provisioning



- Through Web Console
- Through command-line
- Through tools
 - vagrant-aws plugin
 - chef-provisioning
 - Ansible AWS module

What is Vagrant

- Vagrant can create and configure virtual development environments
 - Lightweight
 - Reproducible
 - Portable
 - Wrapper around VirtualBox, KVM, AWS, Docker, VMWare, Hyper-V
- Create identical development environments for Operations and Developers
 - No more “works on my machine”
 - Disposable environments
 - Quickly test things like shell scripts, Chef / Ansible configurations, and isolated deployments of your software products

Getting started

- Vagrant will come out of the box with support for VirtualBox
- To get a fresh Ubuntu 14.04 machine running, use following commands:

```
$ vagrant init ubuntu/trusty64  
$ vagrant up --provider virtualbox
```

- Now ssh in your newly built machine:

```
$ vagrant ssh
```

- When finished using it, you can dispose the machine using:

```
$ vagrant destroy
```

Vagrantfile

- A project has a Vagrantfile with the configuration
 - It marks the root directory of your project
 - It describes the kind of machine and resources needed for the project to run and what software to install
 - The Vagrantfile is meant to be included in version control system
- To create a vagrant file:

```
$ mkdir new_vagrant_project  
$ cd new_vagrant_project  
$ vagrant init
```

Using a box

- To use a box, the Vagrantfile should look like this:

```
Vagrant.configure("2") do |config|
  config.vm.box = "ubuntu/trusty64"
end
```

- `Vagrant.configure("2")` refers to Vagrant version 2 syntax
- Vagrant will automatically download and add a box, if it is downloaded yet

Synced folders

- The project directory is available in the guest machine:

```
vagrant@trusty64:~$ ls /vagrant  
Vagrantfile
```

- Vagrant will automatically sync your files to and from the guest machine
 - Changes that are made in /vagrant are visible for the host system
 - The host machine can be used to edit files in the project folder, rather than using an editor over ssh

Software Provisioning

- Vagrant has support for automated provisioning:

```
Vagrant.configure("2") do |config|
  config.vm.box = "ubuntu/trusty64"
  config.vm.provision :shell, path: "bootstrap.sh"
end
```

- Vagrant will automatically run bootstrap.sh when the guest machine is started
- bootstrap.sh is placed in the project folder and can contain commands to install and configure software

Port Forwarding

- If we want to run services, we can add port forwarding in the Vagrantfile:

```
Vagrant.configure("2") do |config|
  config.vm.box = "hashicorp/precise32"
  config.vm.provision :shell, path: "bootstrap.sh"
config.vm.network :forwarded_port, host: 8080, guest: 80
```

- This will open port 8080 on the host machine and redirect it to port 80 (web) on the guest machine
- After adding port forwarding, “vagrant reload” must be run to reload the Vagrantfile

Shutting down

- Save the current state:

```
$ vagrant suspend
```

- Gracefully shutdown the machine:

```
$ vagrant halt
```

- Power down the machine and remove all hard disks (but preserve Vagrantfile)

```
$ vagrant destroy
```

- The machine can be started again with vagrant up, which will build the machine from the base image, start it again and will run the bootstrap

Vagrant demo

Step 2 - Automation and Configuration Management

Configuration Management Tools

- Puppet
 - Oldest one: initial release: 2005
 - Client-server
- Chef
 - Client-Server
- Ansible
 - Deploys over SSH
- Saltstack
 - Client-Server

Ansible vs. Chef

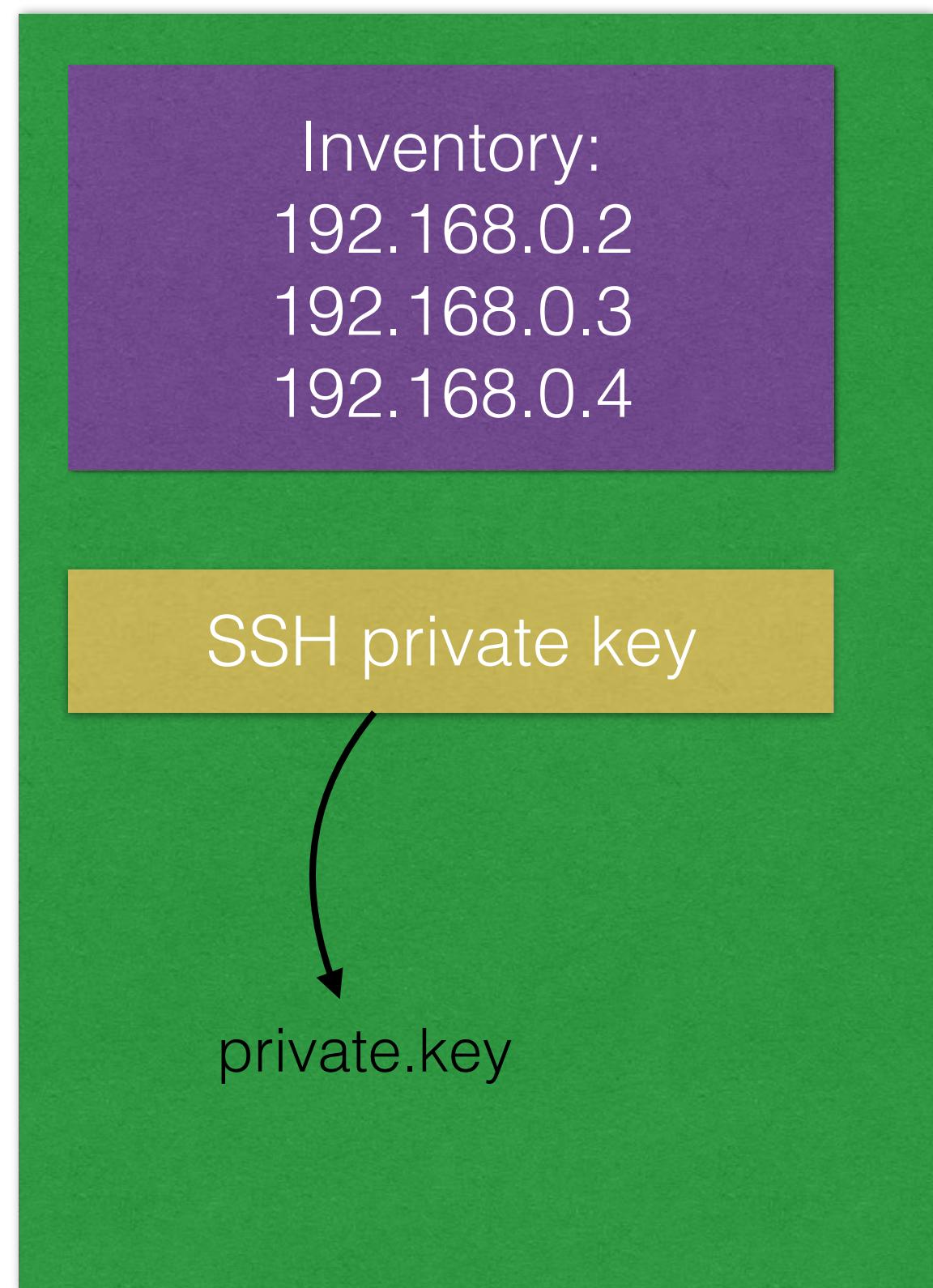
	ANSIBLE	CHEF
ARCHITECTURE	SSH, Minimal in nature	chef-server, chef-client, chef-solo
State	Can be used with Ansible Tower to keep state of servers	Always keeps state in chef-server / hosted chef server
Maturity	Newer	Released in 2009, very mature
Language	Python & YAML	Ruby, ERB & JSON
Learning curve	Low learning curve, easy descriptive language based on YAML	High learning curve
Community	Not as many playbooks available as chef	A lot of recipe's are available
Flexibility	Can be less powerful due to its simplicity	Code driven approach, very flexible
Company	Acquired by Redhat in October 2015	Chef, previously OpsCode (private company, Seattle, Washington)

Ansible

Introduction

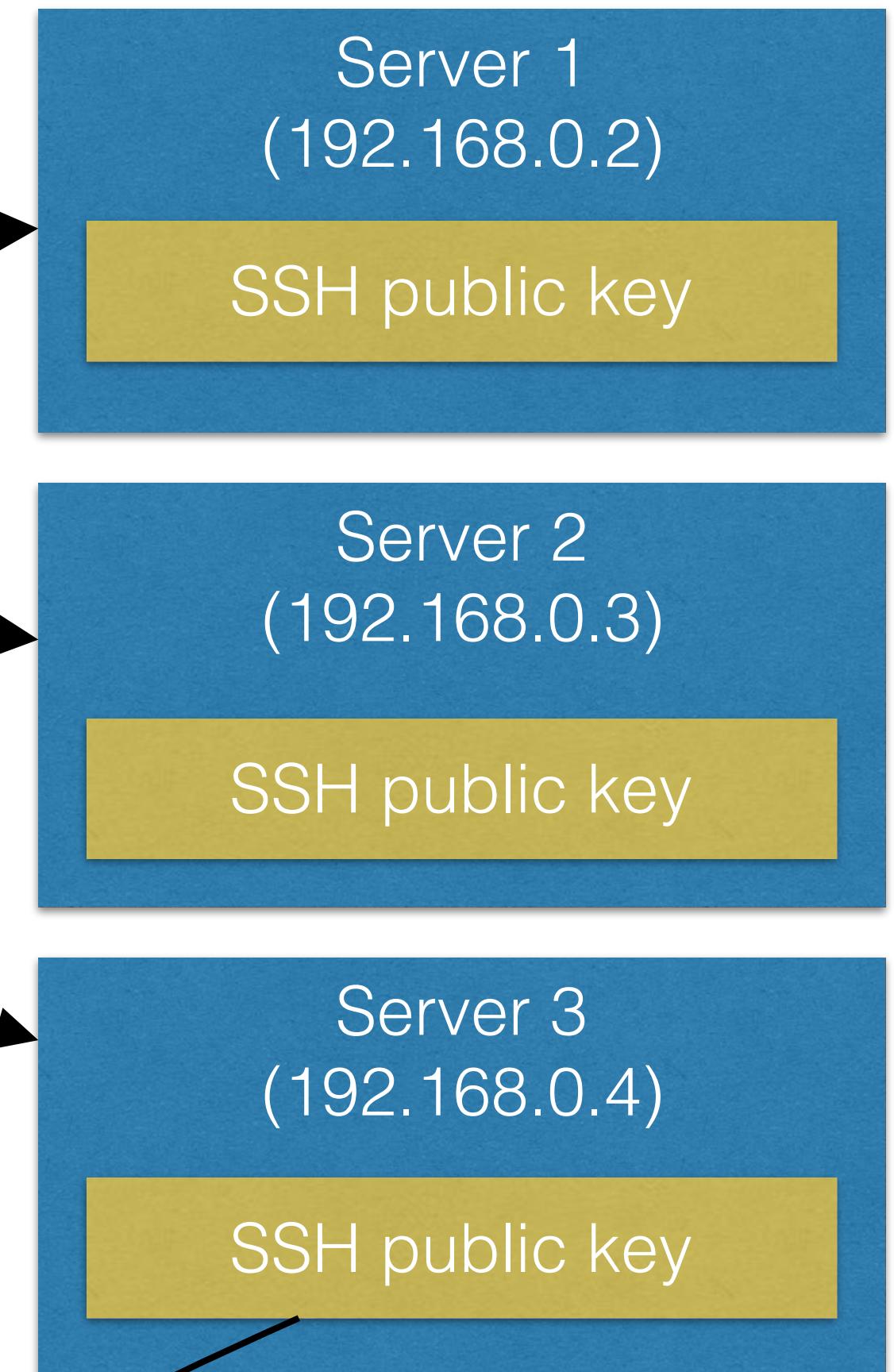
Ansible

Ansible



Ansible connects through
SSH (secure connection) to
the hosts

Hosts



.ssh/authorized_keys

Installing Ansible machine

- Spin up two ubuntu machines
- Create an empty directory somewhere on your system
- Create an empty Vagrantfile with an editor and put the following in it:

```
Vagrant.configure(2) do |config|
  config.vm.define "webserver" do |webserver|
    webserver.vm.box = "ubuntu/trusty64"
    webserver.vm.network "private_network", ip: "192.168.0.2"
    webserver.vm.hostname = "webserver"
  end
  config.vm.define "ansible" do |ansible|
    ansible.vm.box = "ubuntu/trusty64"
    ansible.vm.network "private_network", ip: "192.168.0.254"
    ansible.vm.hostname = "ansible"
  end
```

Installing Ansible machine

- You can now spin up the machines using

```
$ vagrant up ansible  
$ vagrant up webserver
```

- Log to the ansible machine in and install ansible using apt

```
$ vagrant ssh ansible  
vagrant@ansible:~$ sudo apt-get install ansible
```

Installing Ansible machine

- Generate new key pair on the Ansible machine

```
vagrant@ansible:~$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/vagrant/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/vagrant/.ssh/id_rsa.
Your public key has been saved in /home/vagrant/.ssh/id_rsa.pub.
The key fingerprint is:
1a:54:63:2a:e5:fe:78:6e:20:8a:59:90:0a:85:24:ff vagrant@vagrant-ubuntu-trusty-64
```

- private key: /home/vagrant/.ssh/id_rsa (for Ansible machine)
- public key: /home/vagrant/.ssh/id_rsa.pub (to copy to .ssh/authorized_keys on hosts)

Setup host

- Copy the contents of /home/vagrant/.ssh/id_rsa.pub into the clipboard
- Open a new terminal (or cmd) and ssh into our to be webserver:

```
$ vagrant ssh webserver
vagrant@webserver:~$ sudo -s
root@webserver:~# echo 'full contents of id_rsa.pub' > ~/.ssh/authorized_keys
root@webserver:~#
```

Setup Ansible inventory

- To set up the inventory file, we can execute the following commands:

```
$ vagrant ssh ansible  
vagrant@ansible:~$ echo '[webservers]' > hosts  
vagrant@ansible:~$ echo '192.168.0.2' >> hosts
```

- Our inventory file will now look like:

```
[webservers]  
192.168.0.2
```

- We added the IP address of our webserver to the group “webservers” in the inventory file
- In ansible we can define multiple IP addresses or hosts under one group

Setup Ansible inventory

- ssh-agent sends our id_rsa key automatically, so we don't have to put it as an argument

```
vagrant@ansible:~$ ssh-agent bash  
vagrant@ansible:~$ ssh-add .ssh/id_rsa
```

- ansible ping all tests whether all our hosts in our inventory file are reachable

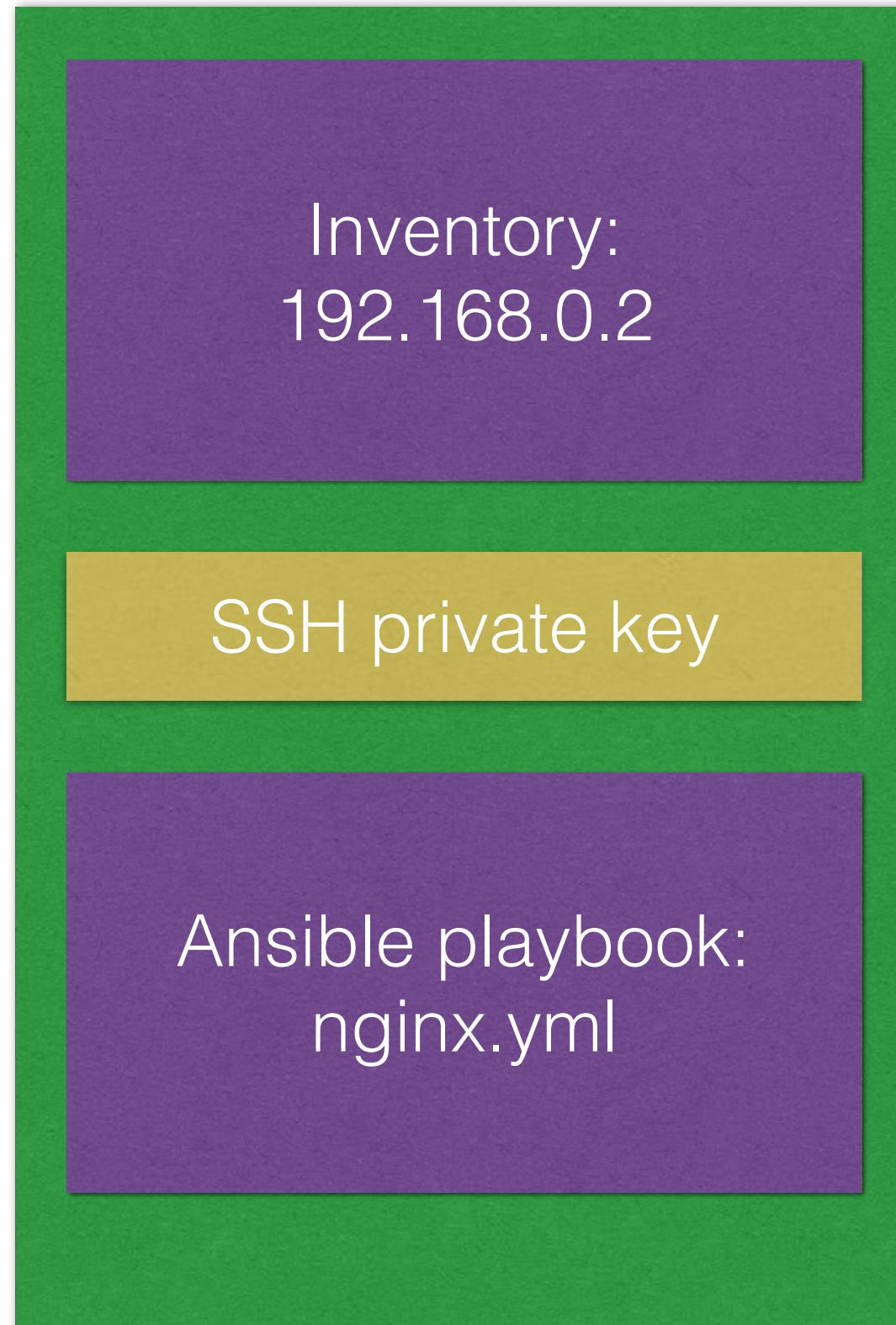
```
vagrant@ansible:~$ ansible -i hosts -u root -m ping all  
192.168.0.2 | success >> {  
  "changed": false,  
  "ping": "pong"  
}
```

Ansible

Installation Demo

Ansible

Ansible



Hosts



Ansible will execute commands
over SSH to install playbooks

Setup Ansible playbook

- Example nginx playbook
 - Only installs nginx, no configuration
- Contents of nginx.yml:

```
---
- hosts: webservers
tasks:
- name: install nginx
  apt: name=nginx state=latest
- name: ensure nginx is running (and enable it at boot)
  service: name=nginx state=started enabled=yes
handlers:
- name: restart nginx
  service: name=nginx state=restarted
```

Run Ansible playbook

- Example nginx playbook (just installs nginx, no configuration)

```
vagrant@ansible:~$ ansible-playbook -i hosts -u root nginx.yml
PLAY [webservers] ****
GATHERING FACTS ****
ok: [192.168.0.2]

TASK: [install nginx] ****
ok: [192.168.0.2]

TASK: [ensure nginx is running (and enable it at boot)] ****
ok: [192.168.0.2]

PLAY RECAP ****
192.168.0.2      : ok=3    changed=1   unreachable=0   failed=0
vagrant@ansible:~$
```

Edit configuration

- Add templates/nginx.conf.j2

```
user {{ user }};
worker_processes {{ worker_processes }};
pid {{ pid }};

events {
    worker_connections {{ worker_connections }} ;
}

http {

    sendfile on;
    tcp_nopush on;
    tcp_nodelay on;
    keepalive_timeout 65;
    types_hash_max_size 2048;
    include /etc/nginx/mime.types;
    default_type application/octet-stream;
    access_log /var/log/nginx/access.log;
    error_log /var/log/nginx/error.log;
    gzip on;
    gzip_disable "msie6";
    include /etc/nginx/conf.d/*.conf;
    include /etc/nginx/sites-enabled/*;
}
```

Edit configuration

- Modify nginx.yml

```
---
```

```
- hosts: webservers
  vars:
    user: www-data
    worker_processes: 2
    pid: /run/nginx.pid
    worker_connections: 768
  tasks:
    - name: install nginx
      apt: name=nginx state=latest
    - name: ensure nginx is running (and enable it at boot)
      service: name=nginx state=started enabled=yes
    - name: write the nginx config file
      template: src=templates/nginx.conf.j2 dest=/etc/nginx/nginx.conf
      notify:
        - restart nginx
    handlers:
      - name: restart nginx
        service: name=nginx state=restarted
```

Ansible

EC2 Provisioning Demo

Ansible

Provisioning

Vagrant Provisioning with Ansible

- It is also possible to run ansible as part of vagrant provisioning
- Configuration changes to Vagrantfile:

```
Vagrant.configure(2) do |config|
  config.vm.define "webserver" do |webserver|
    webserver.vm.box = "ubuntu/trusty64"
    config.vm.network "private_network", ip: "192.168.0.2"
    config.vm.hostname = "webserver"
    ansible.playbook = "nginx.yml"
  end
```

AWS Provisioning with ansible

- Install the boto library and awscli

```
vagrant@ansible:~$ sudo apt-get update && sudo apt-get install python-pip  
vagrant@ansible:~$ sudo pip install boto awscli
```

- You will need an Amazon AWS account (<http://aws.amazon.com>)
- When starting a new one, you can launch a ec2 micro server for free in the first year (free tier usage)

Ansible playbook with boto

- Create a new IAM user in Amazon AWS
- Configure credentials in `~/.aws/credentials`
- Import ssh keypair

```
vagrant@ansible:~$ aws ec2 import-key-pair --key-name ansible --public-key-material file:///.ssh/id_rsa.pub
{
    "KeyName": "ansible",
    "KeyFingerprint": "88:5b:95:a0:5b:49:51:18:84:5c:30:e1:f1:25:8a:86"
}
vagrant@ansible:~$
```

Ansible playbook with boto

- Create a new playbook ec2.yml to provision a new server:

```
- name: Create a new instance
hosts: localhost
gather_facts: False
vars:
  keypair: ansible
  instance_type: t2.micro
  security_group: sg-123456
  image: ami-47a23a30
  region: eu-west-1
  subnet: subnet-123456
  groupname: webservers
tasks:
  - name: Launch instance
    ec2:
      key_name: "{{ keypair }}"
      group: "{{ security_group }}"
```

1

```
  instance_type: "{{ instance_type }}"
  image: "{{ image }}"
  wait: true
  region: "{{ region }}"
  vpc_subnet_id: "{{ subnet }}"
  assign_public_ip: yes
  register: ec2
  - name: Add new instance to host group
    add_host: hostname={{ item.public_ip }}
    groupname={{ groupname }}
    with_items: ec2.instances
  - name: Wait for SSH to come up
    wait_for: host={{ item.public_dns_name }} port=22
    delay=60 timeout=320 state=started
    with_items: ec2.instances
  - include: nginx-ec2.yml
```

2

Ansible playbook with boto

- nginx-ec2.yml modifications
- AWS ubuntu image doesn't allow root login
- .ssh/authorized_keys gets installed in the home of the ubuntu user

```
---
- hosts: webservers
  sudo: yes
  remote_user: ubuntu
  vars:
    user: www-data
    worker_processes: 2
    pid: /run/nginx.pid
    worker_connections: 768
  ...
```

Maintaining inventory

- It's not recommended using a static inventory file when using cloud providers like AWS
- Better is to pass a script to ansible to build the inventory dynamically
- ansible-playbook -i ec2.py will build the inventory dynamically
 - The script will access the AWS API to gather the IP addresses
 - ec2.py can be found on the ansible website or in my GIT repository with all ansible playbook used in this course

Ansible

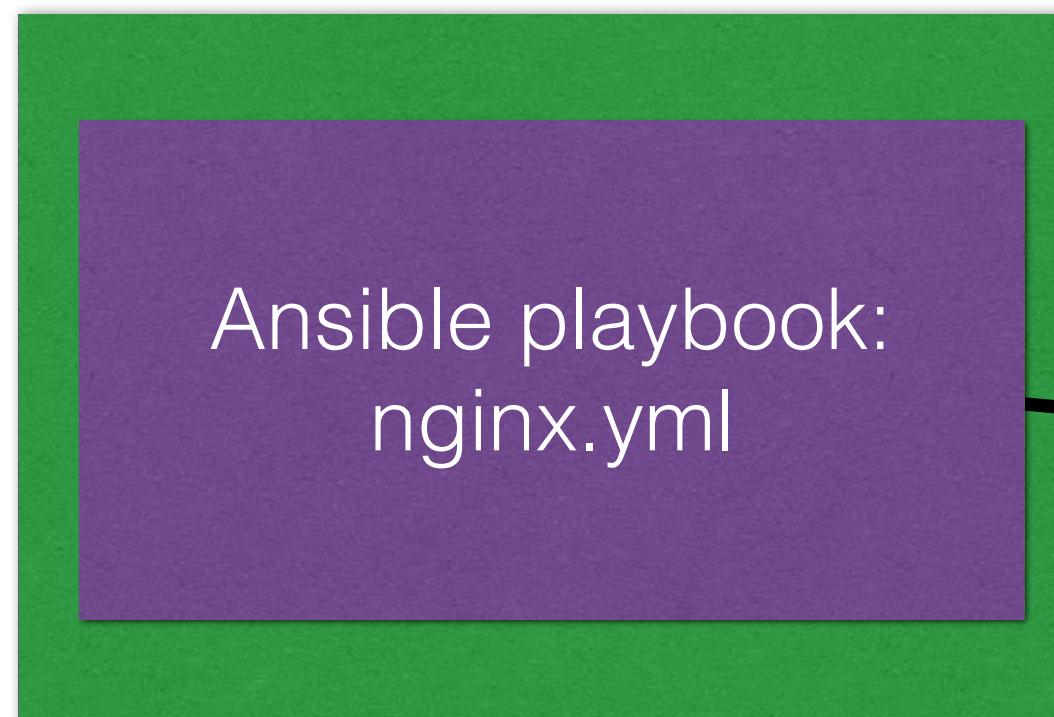
Provisioning demo

Ansible

Roles

Ansible Roles

Ansible



Ansible playbook:
`nginx.yml`

roles/webservers/defaults:
roles/webservers/files:
roles/webservers/handlers:
main.yml
roles/webservers/meta:
roles/webservers/tasks:
main.yml
roles/webservers/templates:
nginx.conf.j2
roles/webservers/vars:
main.yml

Ansible roles

- roles/webservers/vars/main.yml

```
---
```

```
user: www-data
worker_processes: 2
pid: /run/nginx.pid
worker_connections: 768
```

Ansible roles

- roles/webservers/tasks/main.yml

```
---
- name: install nginx
  apt: name=nginx state=latest
- name: ensure nginx is running (and enable it at boot)
  service: name=nginx state=started enabled=yes
- name: write the nginx config file
  template: src=nginx.conf.j2 dest=/etc/nginx/nginx.conf
  notify:
    - restart nginx
```

Ansible roles

- roles/webservers/handlers/main.yml

```
---
- name: restart nginx
  service: name=nginx state=restarted
```

Ansible roles

- webservers.yml

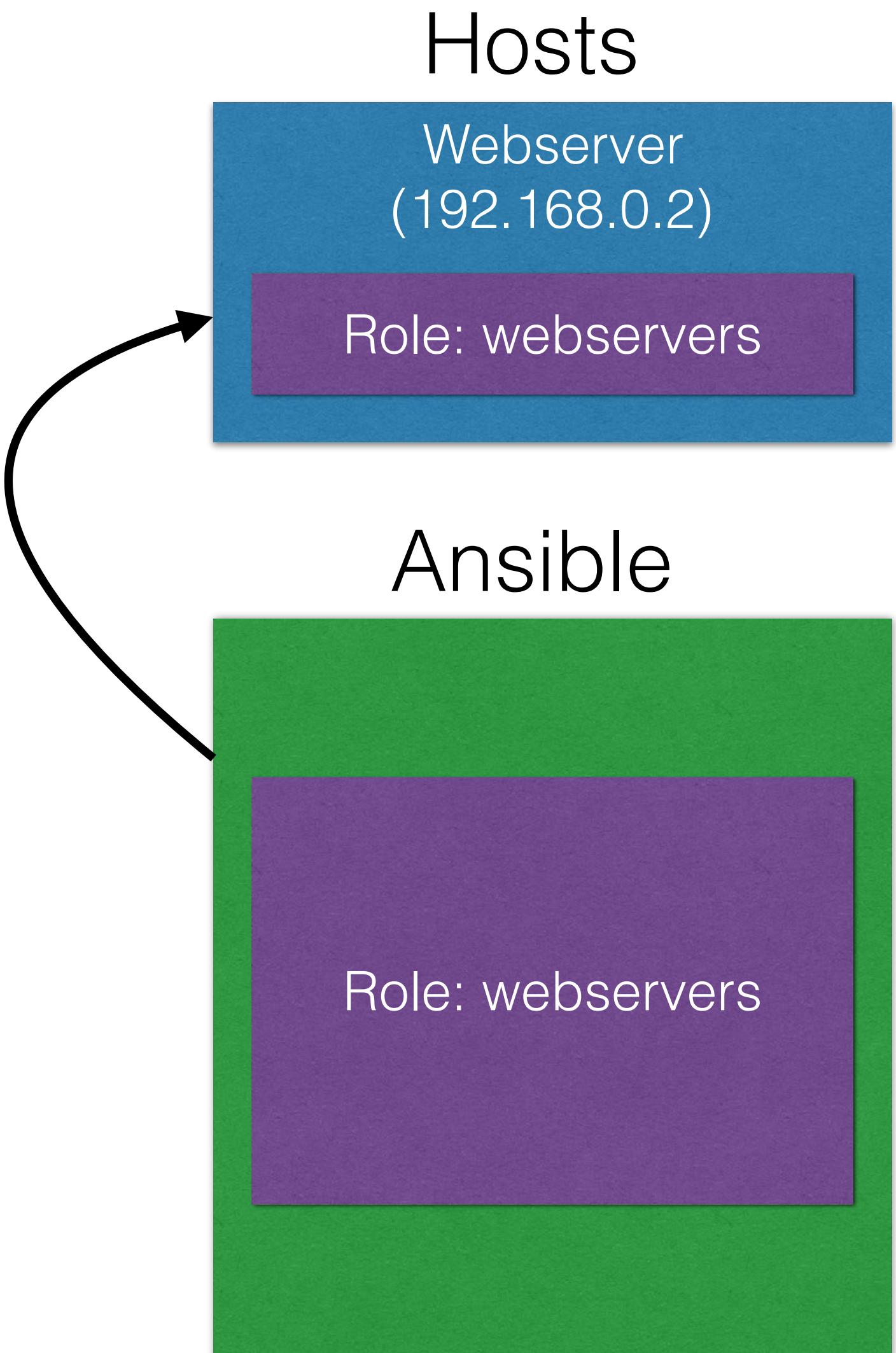
```
---
- hosts: webservers
  roles:
    - role: webservers
```

- Run ansible playbook:

```
vagrant@ansible:~/ansible-demo$ ansible-playbook -i hosts -u root webservers.yml
PLAY [webservers] ****
GATHERING FACTS ****
ok: [192.168.0.2]

TASK: [webservers | install nginx] ****
ok: [192.168.0.2]
...

```



Ansible

Ansible Tower

Ansible Tower

- Ansible's paid product
- Gives you a lot more features
 - States
 - Dashboard
 - Security
 - Support
- Can be quite expensive, there is also an open source alternative, called semaphore

Ansible

Best Practices

Playbooks: best practices

- Use roles! Always use roles
- Use version control (git)
- Dynamic inventory with clouds
- Tag instances with dev / staging / prod, have separate inventory files
 - You can assign variables to groups to organize them
 - A host can be in multiple groups (e.g. geography + function)
- Use rolling updates (start updating a percentage of your prod servers)

Playbooks: best practices

- Use group_by to deal with different operating systems:

```
---  
# talk to all hosts just so we can learn about them  
- hosts: all  
  tasks:  
    - group_by: key=os_{{ ansible_distribution }}  
# now just on the CentOS hosts...  
- hosts: os_CentOS  
  gather_facts: False  
  tasks:  
    - # tasks that only happen on CentOS go here
```

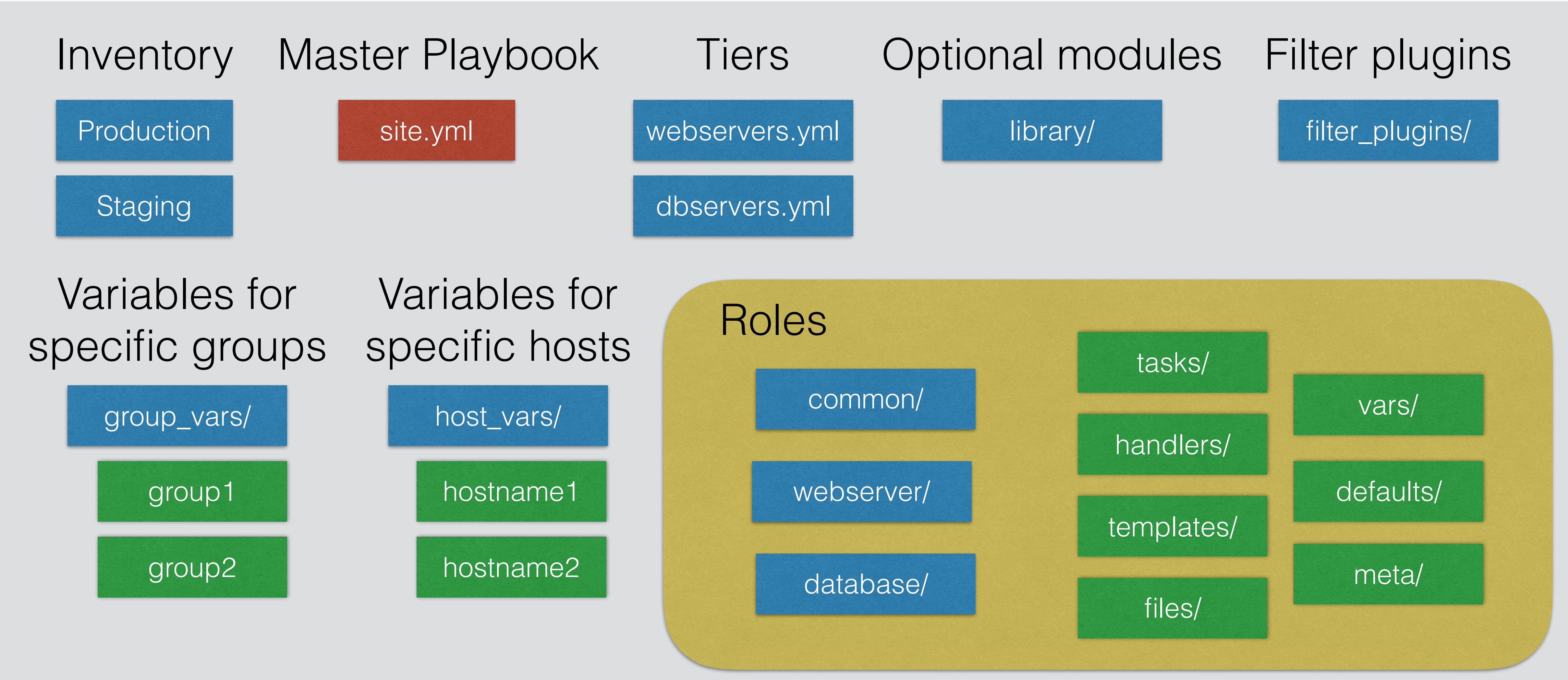
- or split the variables per OS

```
- hosts: all  
  tasks:  
    - include_vars: "os_{{ ansible_distribution }}.yml"
```

Playbooks: best practices

- Use whitespace and comments (#)
- Always name tasks (using '-name')
- Keep it simple!

Playbooks: Directory Layout



Playbooks: Directory Layout

Master Playbook

site.yml

```
---
- include: webservers.yml
- include: dbservers.yml
- include deploy_app.yml
```

Tiers

webservers.yml

```
---
- hosts: webservers
  roles:
    - common
    - webserver
```

dbservers.yml

deploy_app.yml

```
---
# file: deploy_app.yml
- hosts: webservers
  roles:
    - myapp
```

Example commands

- Run full playbook on production hosts

```
$ ansible-playbook -i production site.yml
```

- Only reconfigure webservers

```
$ ansible-playbook -i production webservers.yml
```

- Only reconfigure webservers in a certain group

```
$ ansible-playbook -i production webservers.yml --limit group
```

Where to find Ansible content

- Often have to write your own ones
- Enter: “ansible playbook + software name” in search engine
 - Often github repositories
 - See how many people favorited or forked the repository to see how popular it is
 - Might not always be tested with the latest releases
- Ansible Galaxy: <https://galaxy.ansible.com/>

Chef

What is Chef

- Chef is an automation platform for developers & systems engineers to continuously define, build, and manage infrastructure
- Infrastructure as code
 - Programmatically provision and configure
 - Treat like any other codebase
 - Reconstruct business from code repository, data backup, and bare metal resources
- Learning chef is like learning the basics of a language
 - The best way to learn chef is to use chef

Ways of running chef

- Hosted chef
 - Chef server as a Service
 - You pay for usage
 - Chef server
 - In-house chef server (more complex)
 - Free when using the open source version
 - AWS Opsworks
 - Free, if using to provision EC2 machines
 - Useful only if you are already AWS customer or planning to be

Chef Terminology

- Organizations
- Environments
- Roles
- Nodes
- Recipes
- Cookbooks
- Search

Organizations

- Completely independent tenants
- Share nothing with other organizations
- May represent different
 - Companies
 - Business Units
 - Departments



Environments

- Typically Development, QA / Staging, Production
- Life stages of the applications
- Environments policies may include data attributes necessary for configuring infrastructure
 - Location of a package repository
 - Version of chef configuration files

Organization X

Development

Staging

Production

Nodes

- Represent the servers
- Can be physical or virtual
- Each node belongs to one environment in one organization
- Each node has zero or more Roles

Organization X

Production

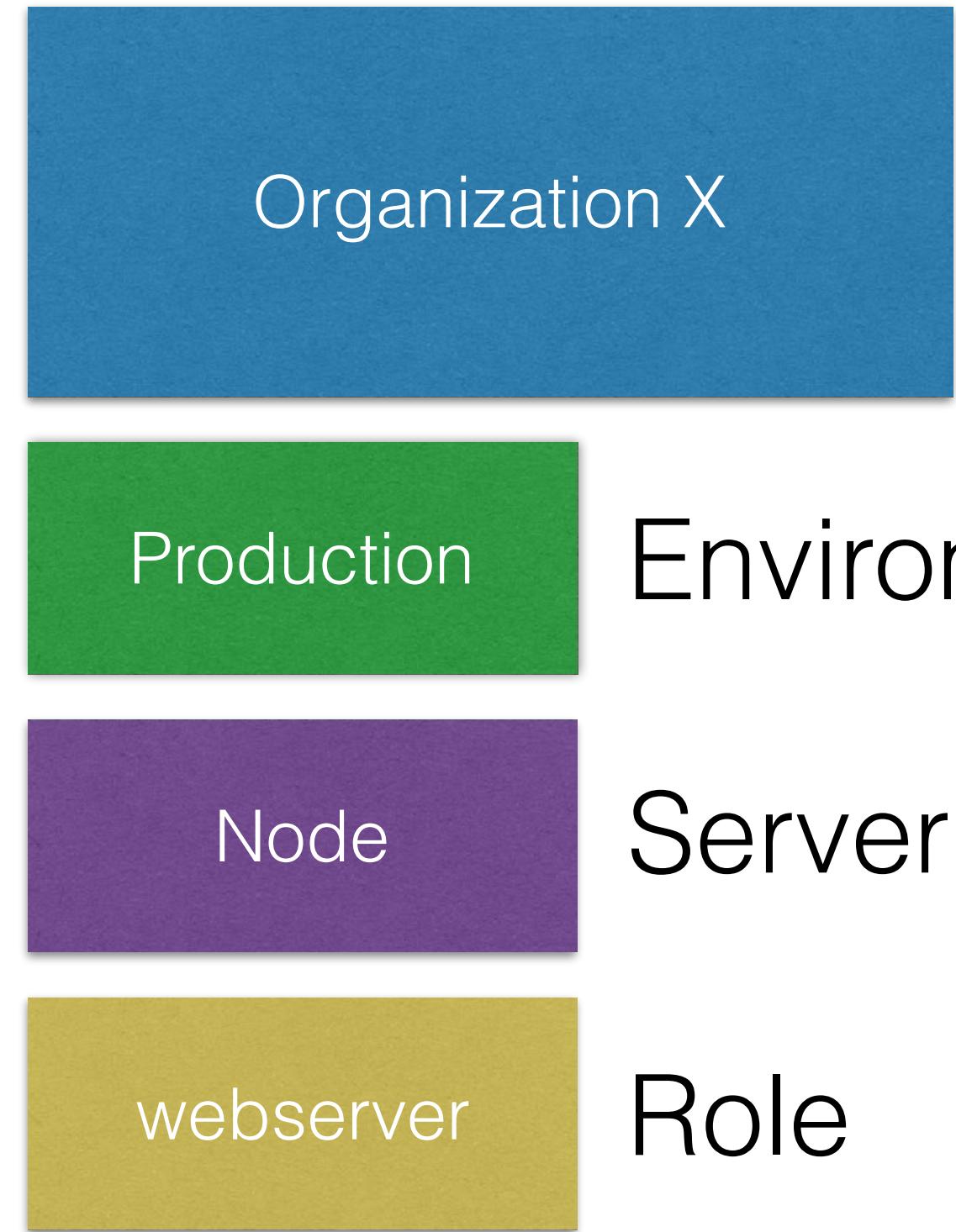
Node

Environment

Server

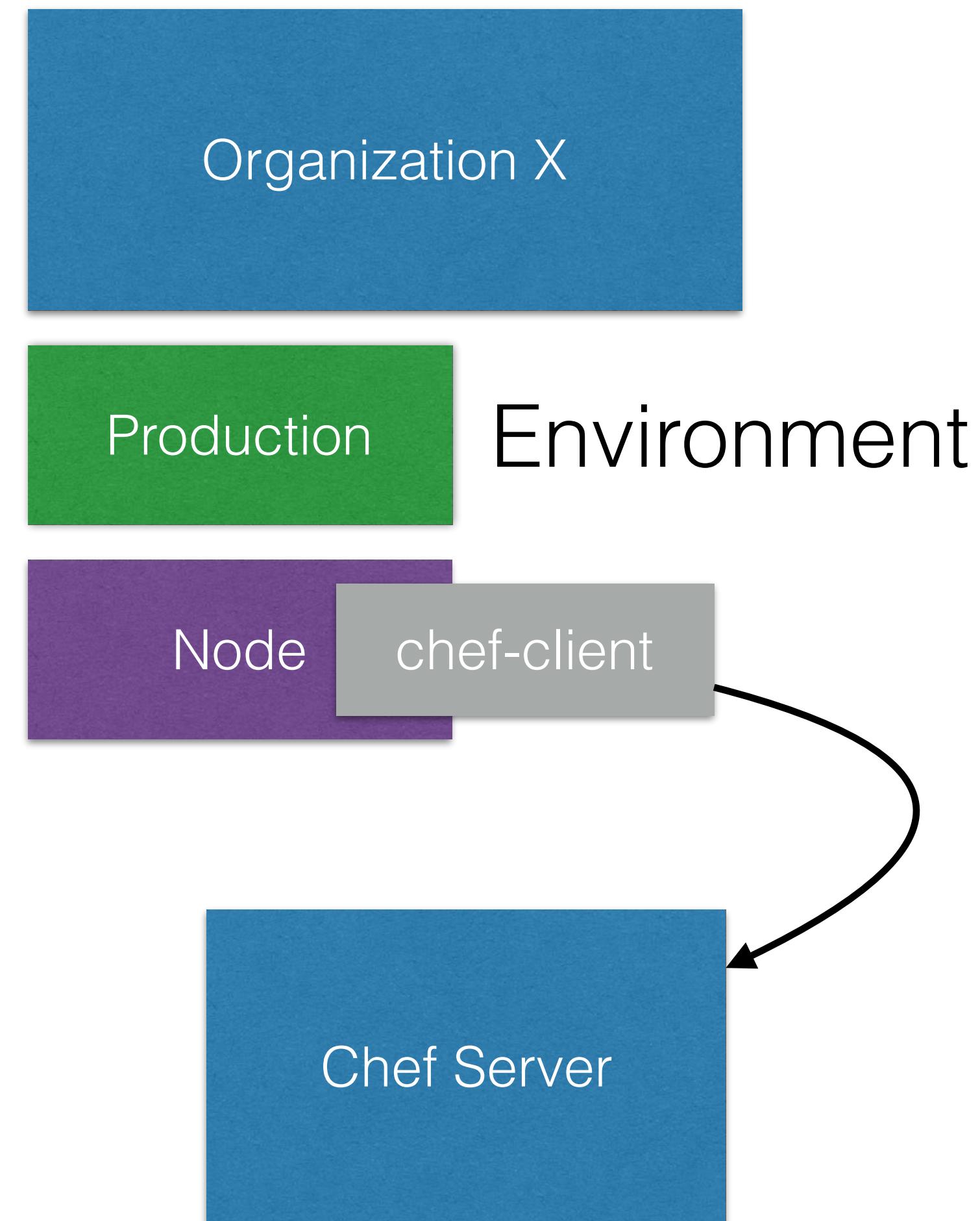
Roles

- Roles represent the types of servers
 - Database Server
 - VPN Server
 - Application Server
 - Monitoring Server
- Roles typically include
 - A run list: list of chef configuration files to be applied
 - Example: Role VPN may include 'openvpn' in the run list
 - Data attributes: attributes necessary to configure the infrastructure
 - Example: OpenVPN IP, Port, other settings



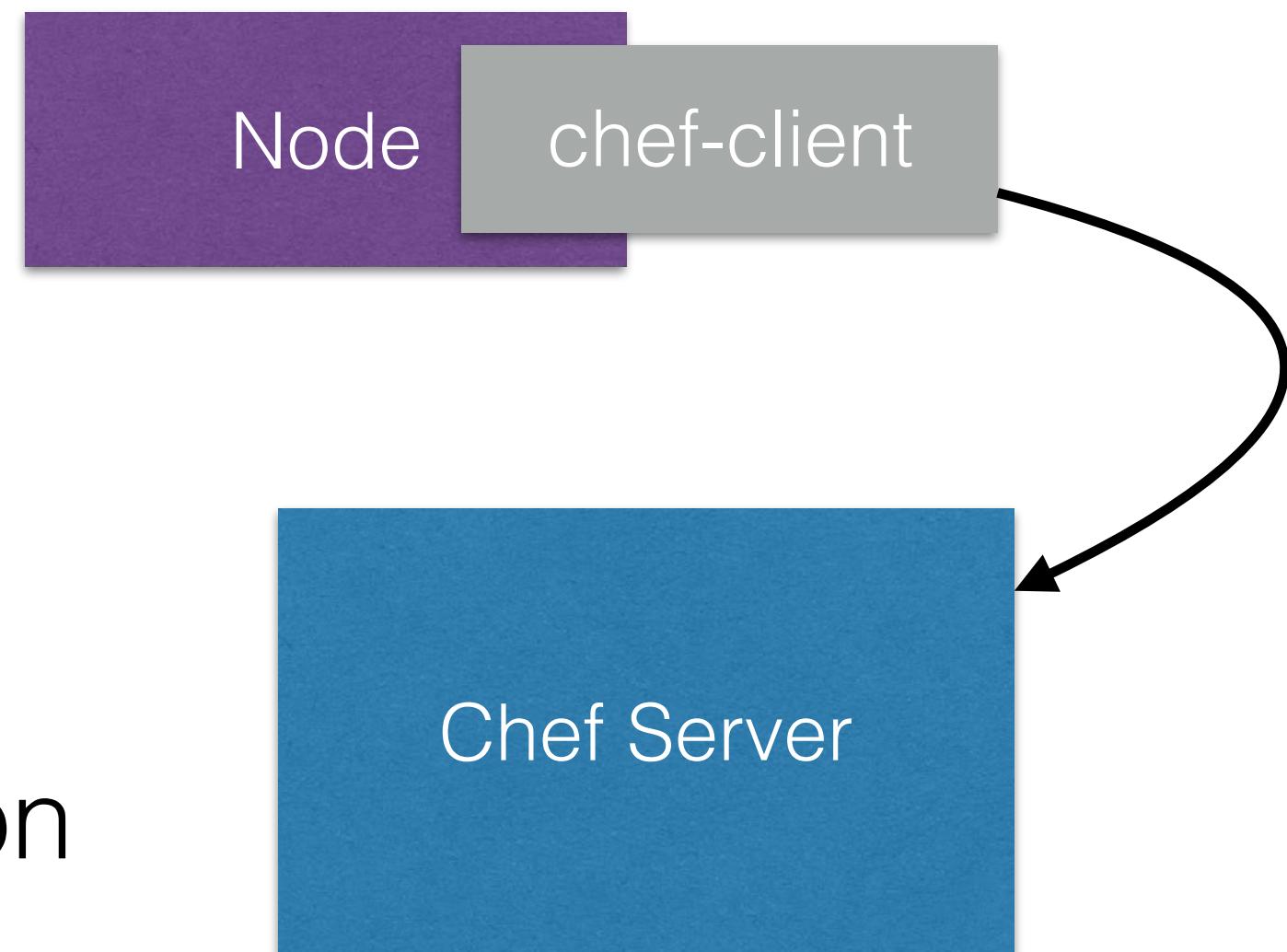
Chef-client

- Chef-client is an application that runs on each node
 - Gathers current system configuration
 - Downloads desired system configuration from Chef Server
 - Configures every node so its state matches the policy described by Chef Server



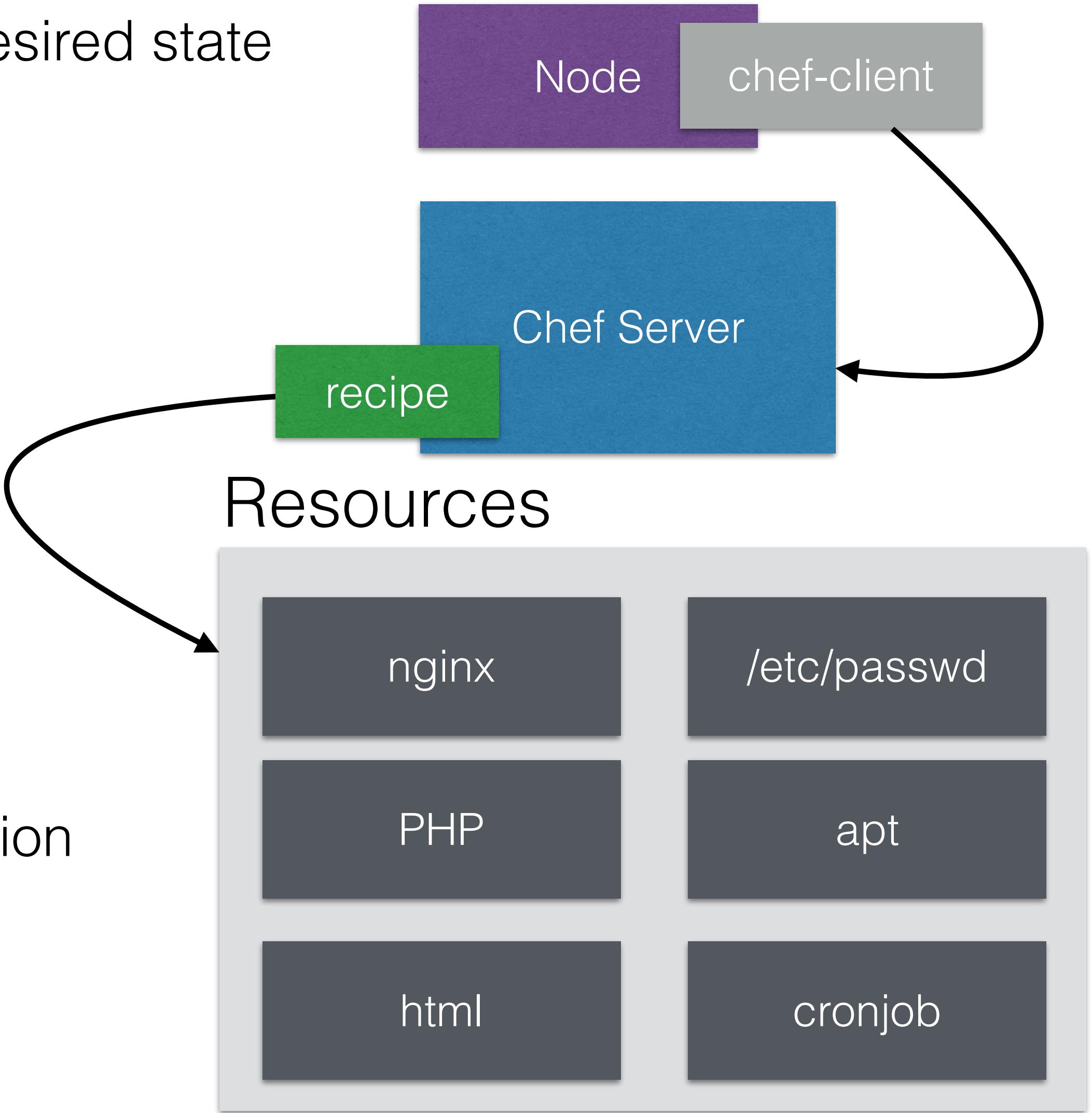
Node configuration

- Chef ensures each node complies with the policy
 - Chef-client will pull the policy from the chef server and enforce policy on that node
 - Policy describes what state each resource should be in
 - Policy is determined by configurations in the run list
- Goal is to reduce management complexity through abstraction
 - Infrastructure as code
 - Store the configuration of your infrastructure in version control



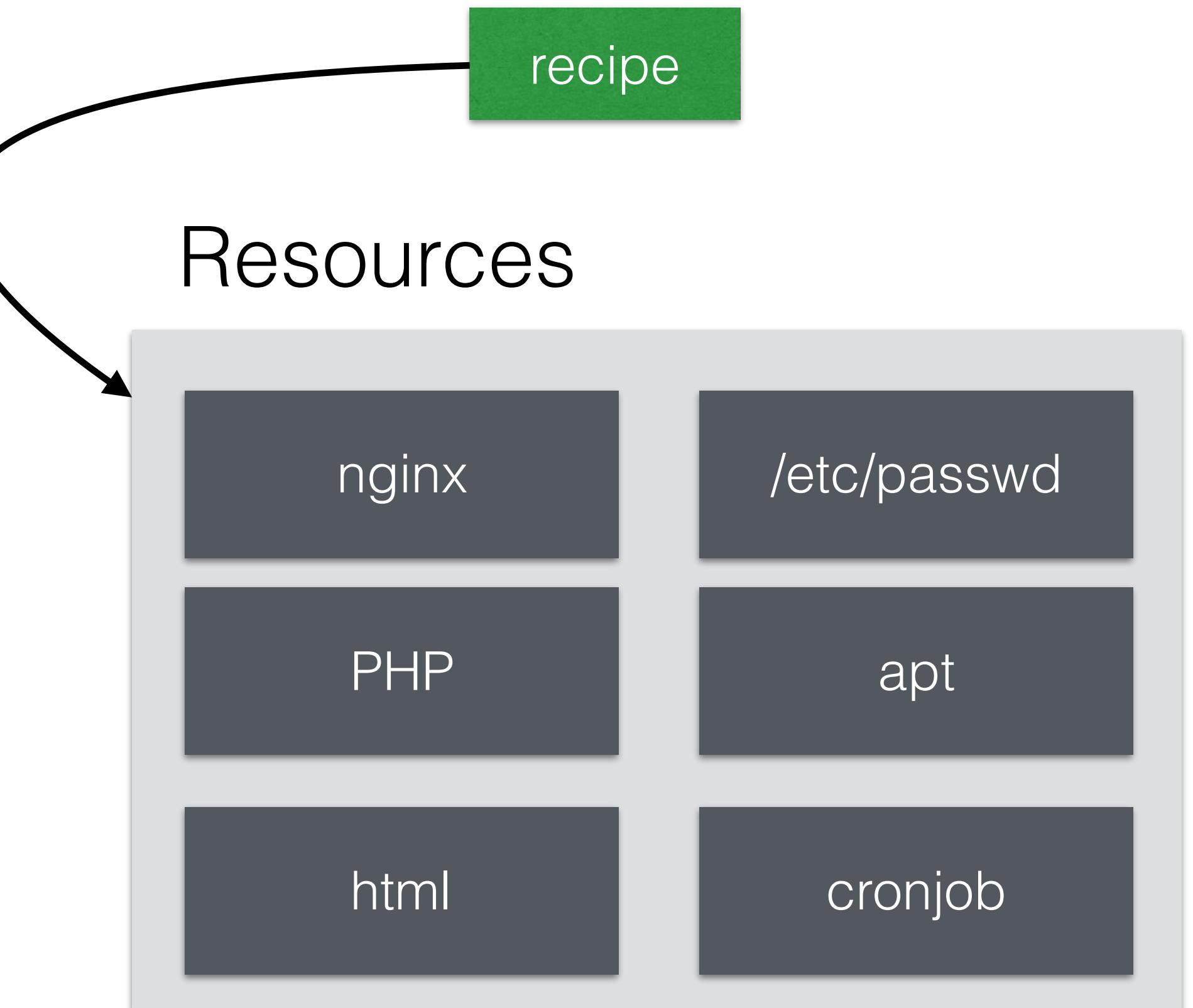
A resource

- A resource represents a piece of a system and its desired state
- Examples:
 - A package that needs to be installed
 - A service that needs to be running
 - A user that needs to be present
 - A scheduled task that needs to be configured
- Resources are the building blocks of chef configuration
- Resources are included in recipes
 - Recipes ensure the system is in a desired state



Recipes

- Recipes are configuration files that describe resources and their desired state
 - Install and configure software components
 - Manage files
 - Deploy applications
 - Can depend on other recipes



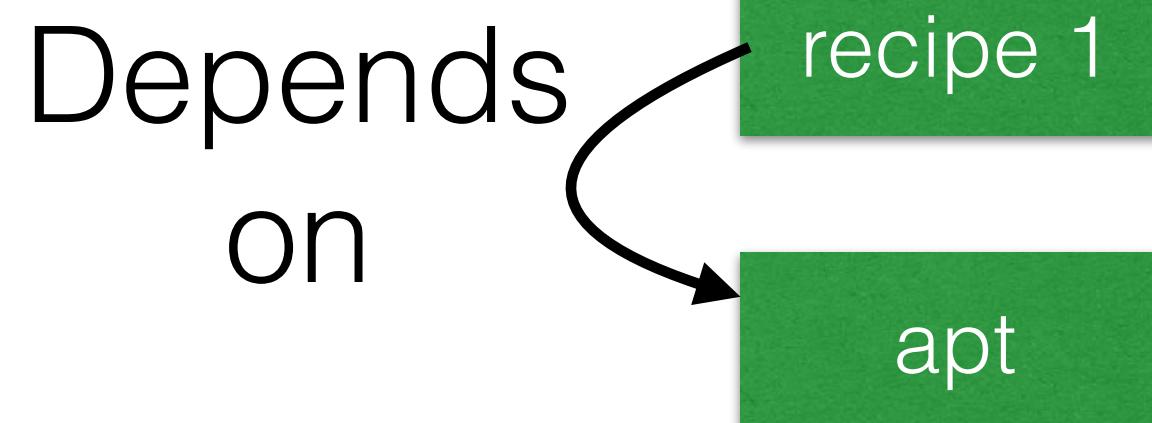
Example recipe

- Install and start nginx
- It uses an existing recipe ‘apt’ to take care of package management

```
include_recipe "apt"

package 'nginx' do
  action :install
end

service 'nginx' do
  action [ :enable, :start ]
end
```



Cookbooks

- Recipes are stored in cookbooks
- Cookbooks contain recipes, templates, files, custom resources, etc
- Provides code re-use and modularity
- Cookbooks can be shared and downloaded from a marketplace
 - A lot of cookbooks are available for popular software
 - Cookbooks that depend on existing cookbooks can customize existing cookbooks

Cookbook 1

recipe 1

recipe 2

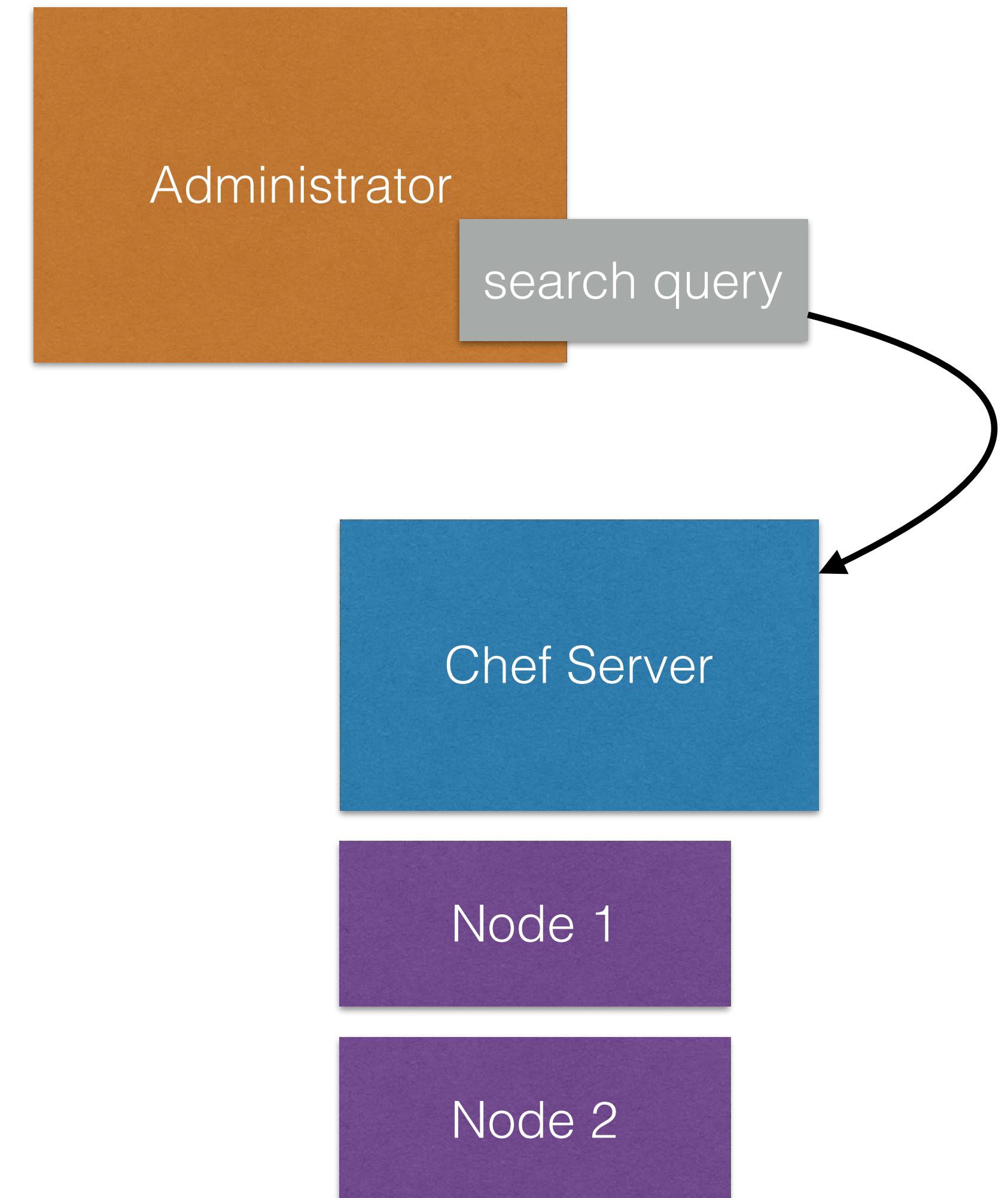
templates

files

...

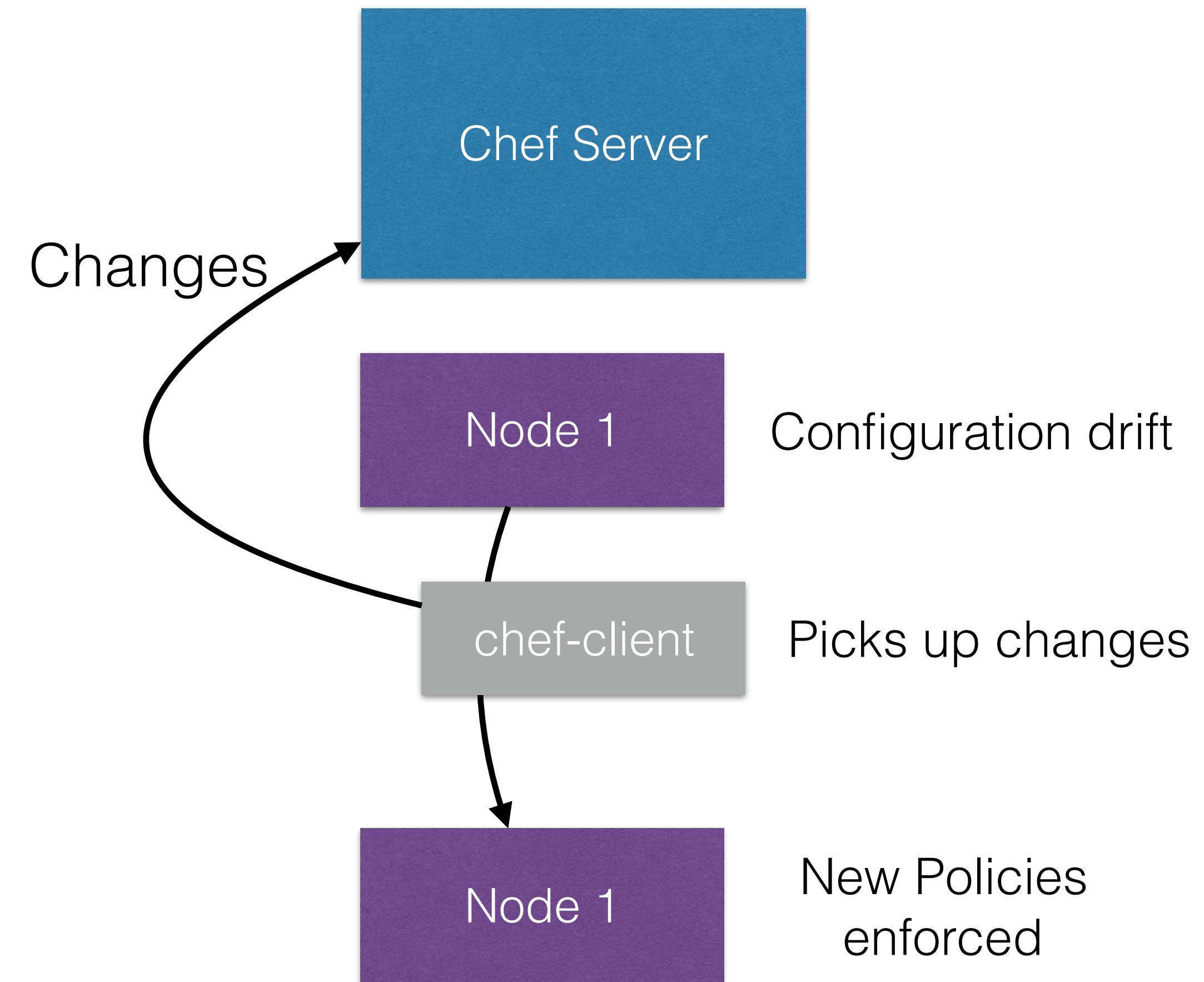
Search

- Search for nodes with roles
- Find topology data (IP address, hostname, Fully Qualified Domain Name)



Configuration Drift

- Configuration Drift happens when:
 - Your infrastructure requirements change
 - The configuration of a server falls out of policy
 - You will need to model the new requirements in your chef configuration files
 - chef-client picks up the changes
 - New policies are enforced



Chef knife

Setting up knife

- New Vagrantfile
- Create a new project folder called chef and create Vagrantfile

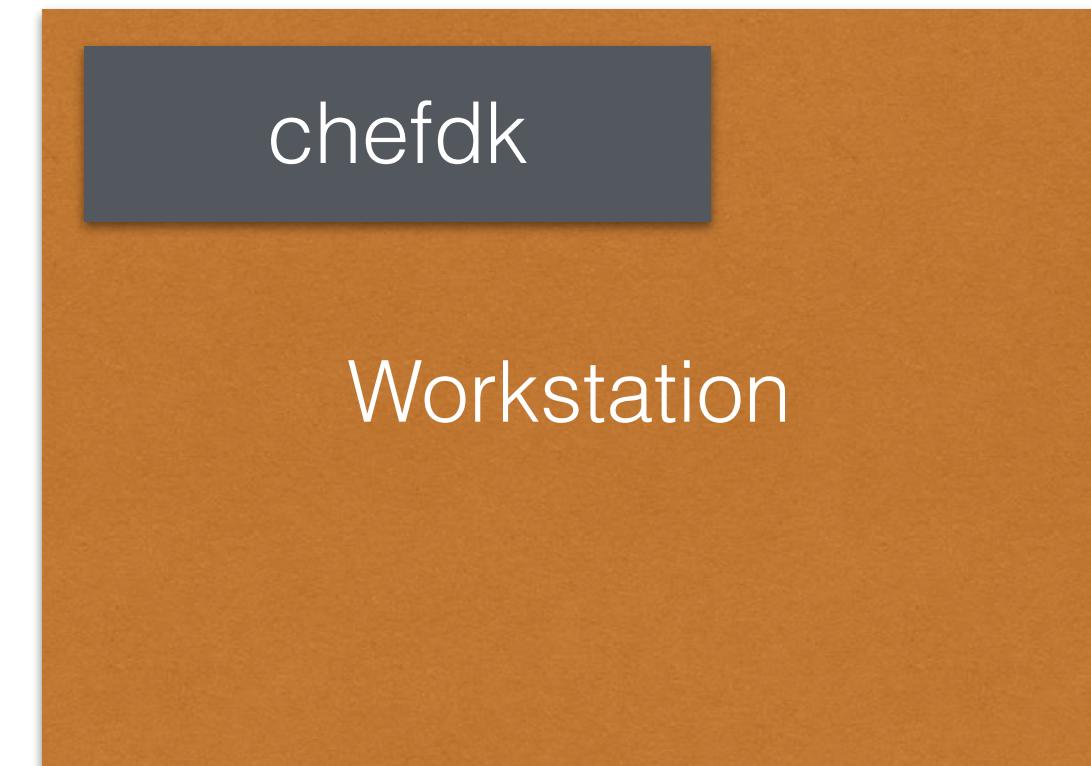
```
Vagrant.configure(2) do |config|
  config.vm.define "workstation" do |workstation|
    workstation.vm.box = "ubuntu/trusty64"
    workstation.vm.network "private_network", ip: "192.168.0.252"
    workstation.vm.hostname = "workstation.example.com"
  end
end
```

```
$ vagrant up workstation
$ vagrant ssh workstation
```

Chef Development toolkit

- Let's install the chef development toolkit
- Download it from chef.io and move it into the vagrant project folder
- Install it with dpkg -i:

```
vagrant@workstation:~$ sudo -s
root@workstation:~# dpkg -i /vagrant/chefdk_0.9.0-1_amd64.deb
Selecting previously unselected package chefdk.
(Reading database ... 60726 files and directories currently installed.)
Preparing to unpack .../chefdk_0.9.0-1_amd64.deb ...
Unpacking chefdk (0.9.0-1) ...
Setting up chefdk (0.9.0-1) ...
Thank you for installing Chef Development Kit!
root@workstation:~#
```



Chef's Knife

- Knife needs to be set up first
 - Knife is a tool that comes with the Chef Development kit

~/.chef/knife.rb

Workstation

- Create ~/.chef/knife.rb and add

```
cookbook_path [ '/home/vagrant/cookbooks' ]
```

- Once you start using Chef Server or a hosted chef server, this knife.rb can be autogenerated and downloaded
 - It contains all information to connect to the Chef Server

Creating first cookbook

- To start writing recipes, create a cookbook directory structure in the folder called my_cookbook:

```
vagrant@workstation:~$ mkdir cookbooks  
vagrant@workstation:~$ cd cookbooks  
vagrant@workstation:~/cookbooks$ chef generate cookbook my_cookbook
```

my_cookbook

metadata

readme

recipe

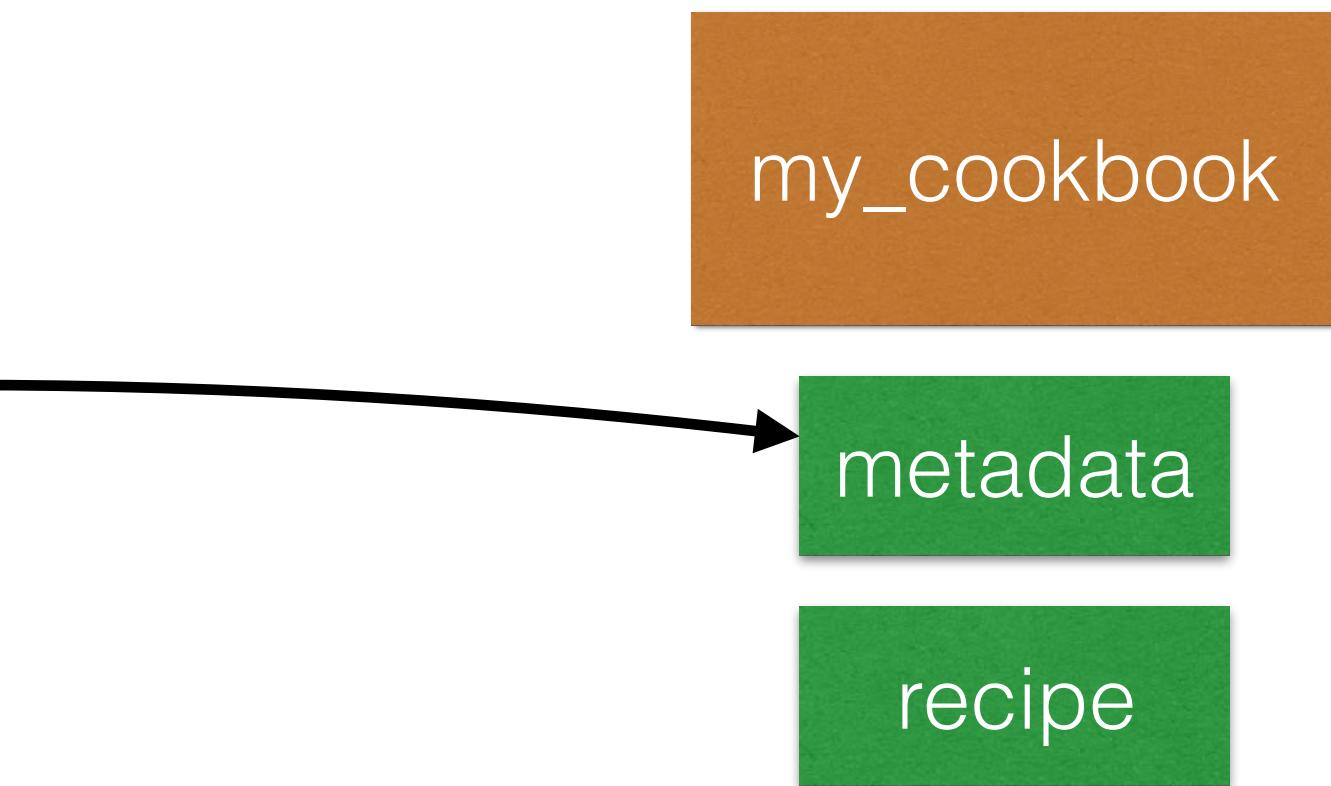
...

```
my_cookbook/  
    ├── Berksfile  
    ├── chefignore  
    ├── metadata.rb  
    ├── README.md  
    └── recipes/  
        └── default.rb
```

Cookbook Metadata

- First, edit the metadata in my_cookbook/metadata.rb

```
name 'my_cookbook'  
maintainer 'Company Inc'  
maintainer_email 'cookbooks@company.com'  
license 'Apache 2.0'  
description 'Installs and configures an example'  
version '0.1.0'  
depends 'apt'
```



The recipe

- The actual recipe resides in my_cookbook/recipes/default.rb:

```
include_recipe "apt"

package 'nginx' do
  action :install
end

service 'nginx' do
  action [:start, :enable]
end

file '/usr/share/nginx/html/index.html' do
  content '<html>
<body>
  <h1>hello world</h1>
</body>
</html>'
end
```

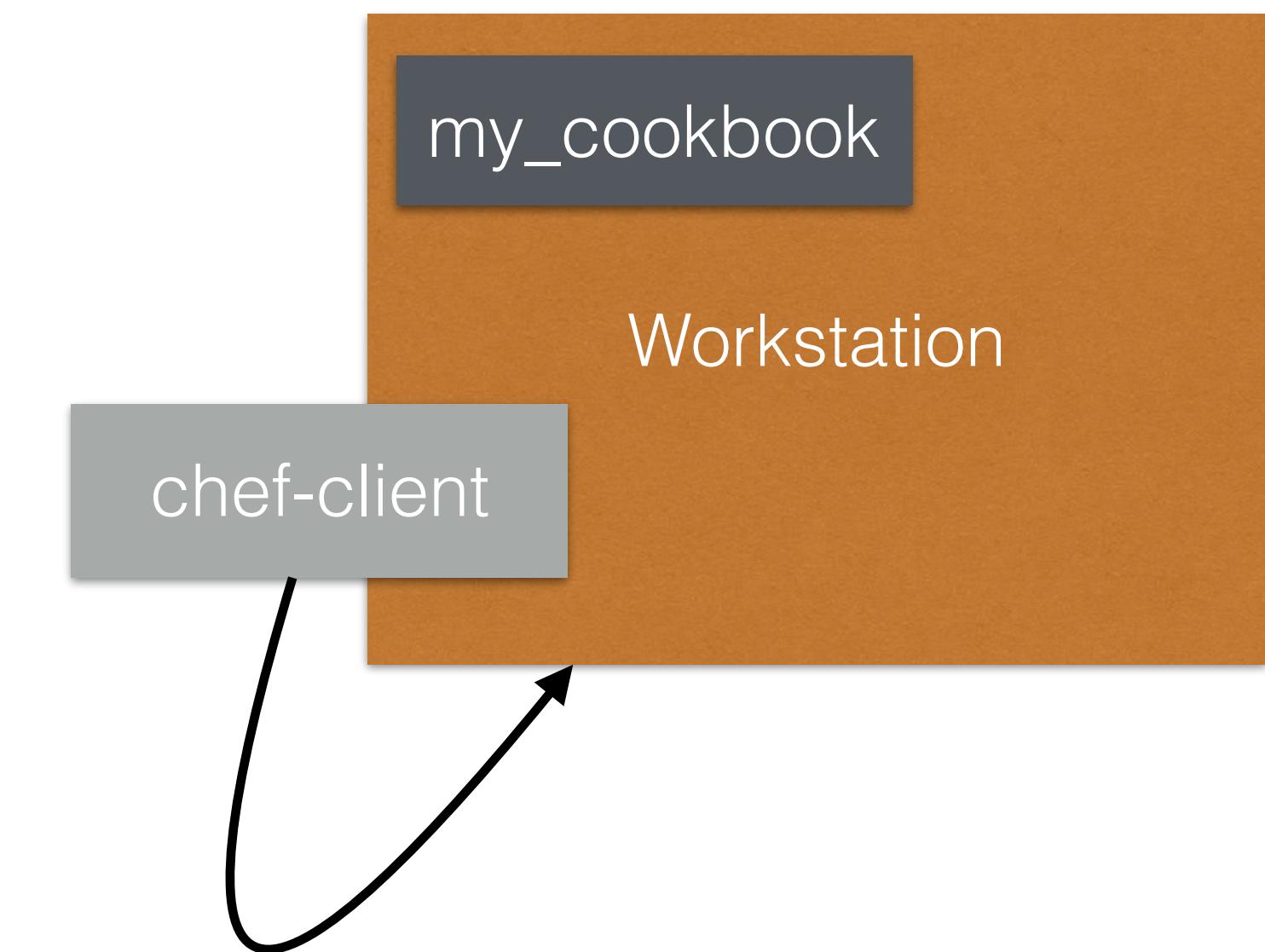


Applying the recipe locally

- Chef-client can be used to execute a cookbook locally:

```
vagrant@workstation:~$ sudo chef-client -z --runlist 'recipe[my_cookbook]'
```

- The cookbook will be executed on the local machine
- An easy way to test cookbooks without the need of chef-server



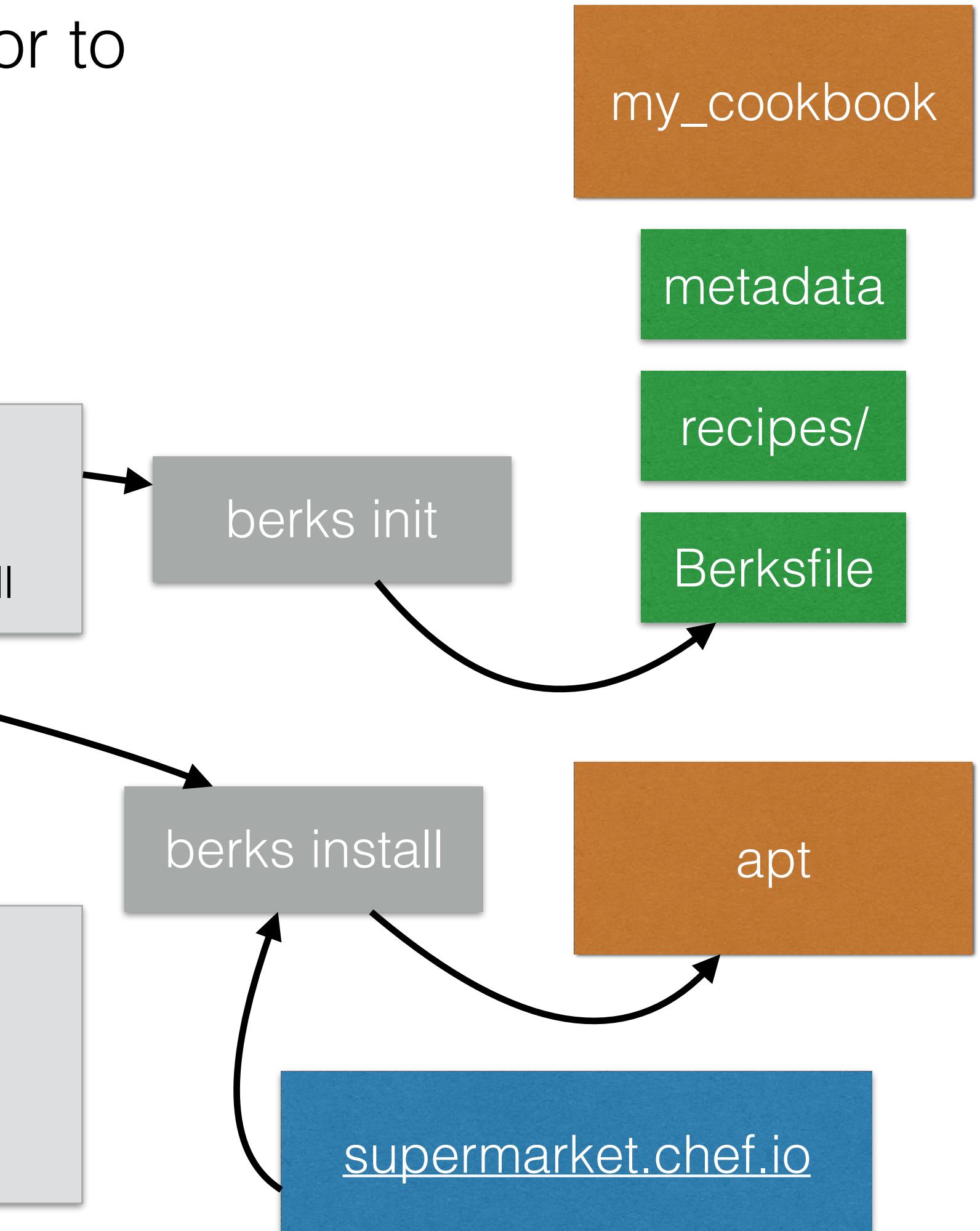
Package recipe

- Package the cookbook for standalone usage, to distribute, or to download all the dependencies
- To initialise berks and download all dependencies:

```
vagrant@workstation:~/cookbooks/my_cookbook$ berks init  
vagrant@workstation:~/cookbooks/my_cookbook$ echo 'cookbook "apt"' >> Berksfile  
vagrant@workstation:~/cookbooks/my_cookbook$ BERKSHelf_PATH=~/cookbooks berks install
```

- To package all cookbooks for distribution:

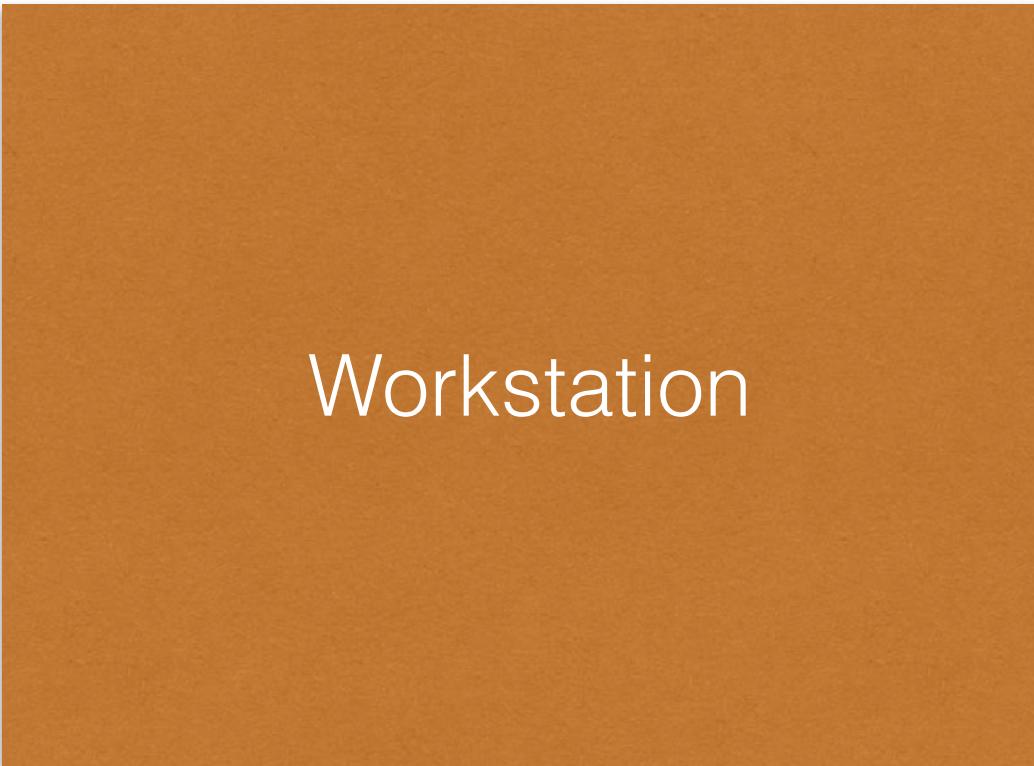
```
vagrant@workstation:~/cookbooks/my_cookbook$ berks vendor  
vagrant@workstation:~/cookbooks/my_cookbook$ ls berks-cookbooks/  
apt my_cookbook
```



Chef knife demo

knife with chef-server

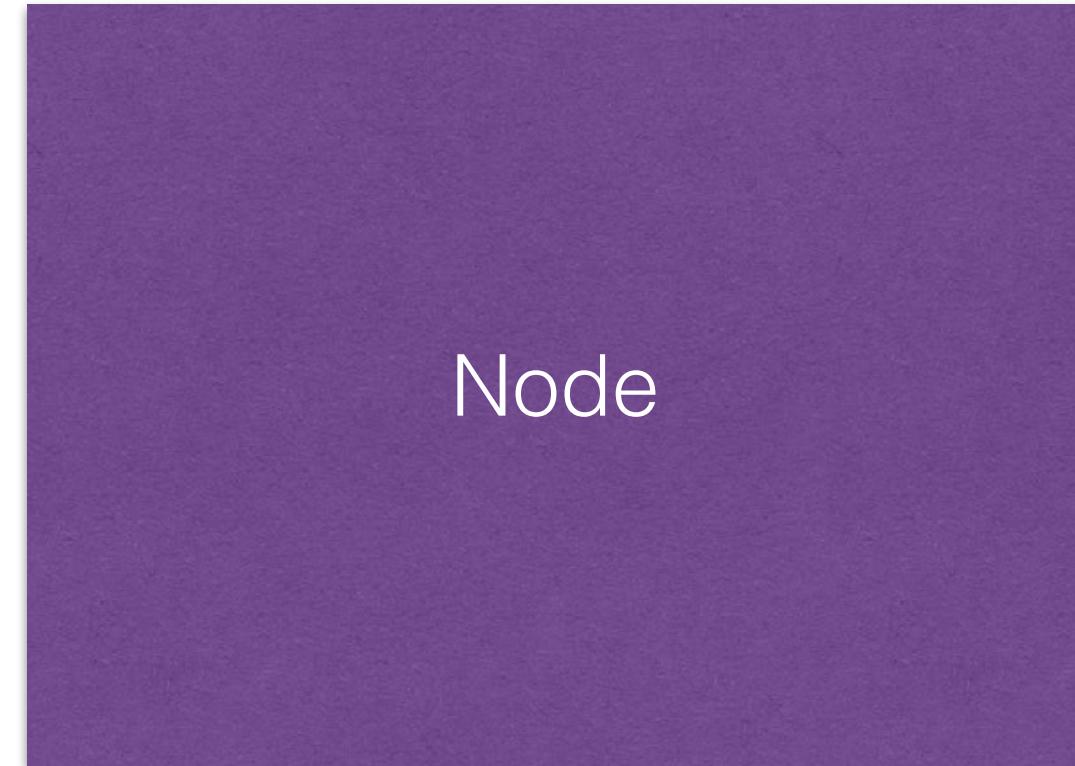
Setting up chef server



Workstation



Chef Server



Node

- Your workstation is the computer from which you create your cookbooks and administer your network
- Chef server acts as a central repository for your cookbooks as well as for information about every node it manages.
- A node is any computer that is managed by a Chef server.

Setting up chef-server

```
Vagrant.configure(2) do |config|
  config.vm.define "workstation" do |workstation|
    workstation.vm.box = "ubuntu/trusty64"
    workstation.vm.network "private_network", ip: "192.168.0.252"
    workstation.vm.hostname = "workstation.example.com"
  end
  config.vm.define "node" do |node|
    node.vm.box = "ubuntu/trusty64"
    node.vm.network "private_network", ip: "192.168.0.3"
    node.vm.hostname = "node.example.com"
  end
  config.vm.define "chef" do |chef|
    chef.vm.box = "ubuntu/trusty64"
    chef.vm.network "private_network", ip: "192.168.0.253"
    chef.vm.hostname = "chef.example.com"
    chef.vm.provider "virtualbox" do |v|
      v.memory = 4096
      v.cpus = 2
    end
  end
end
```

Vagrantfile

```
$ vagrant up node
$ vagrant up chef
$ vagrant ssh chef
```

FQDNs

- Chef requires FQDNs (Fully Qualified Domain Names)
- We can fake them by adding it to /etc/hosts:

```
$ vagrant ssh workstation  
vagrant@workstation:~$ sudo -s  
root@workstation:~# echo "192.168.0.253 chef.example.com chef" >> /etc/hosts  
root@workstation:~# echo "192.168.0.252 workstation.example.com workstation" >> /etc/hosts  
root@workstation:~# echo "192.168.0.3 node.example.com node" >> /etc/hosts
```

192.168.0.252

workstation

```
$ vagrant ssh chef  
vagrant@chef:~$ sudo -s  
root@chef:~# echo "192.168.0.253 chef.example.com chef" >> /etc/hosts  
root@chef:~# echo "192.168.0.252 workstation.example.com workstation" >> /etc/hosts  
root@chef:~# echo "192.168.0.3 node.example.com node" >> /etc/hosts
```

192.168.0.253

Chef Server

```
$ vagrant ssh node  
vagrant@node:~$ sudo -s  
root@node:~# echo "192.168.0.253 chef.example.com chef" >> /etc/hosts  
root@node:~# echo "192.168.0.252 workstation.example.com workstation" >> /etc/hosts  
root@node:~# echo "192.168.0.3 node.example.com node" >> /etc/hosts
```

192.168.0.3

Node

Chef-server

- Let's install chef-server
- Download chef-server from chef.io and move it to the vagrant project folder

```
root@chef:~# dpkg -i /vagrant/chef-server-core_12.2.0-1_amd64.deb
Selecting previously unselected package chef-server-core.
(Reading database ... 108086 files and directories currently installed.)
Preparing to unpack .../chef-server-core_12.2.0-1_amd64.deb ...
Unpacking chef-server-core (12.2.0-1) ...
Setting up chef-server-core (12.2.0-1) ...
root@chef:~#
```

192.168.0.253

Chef Server

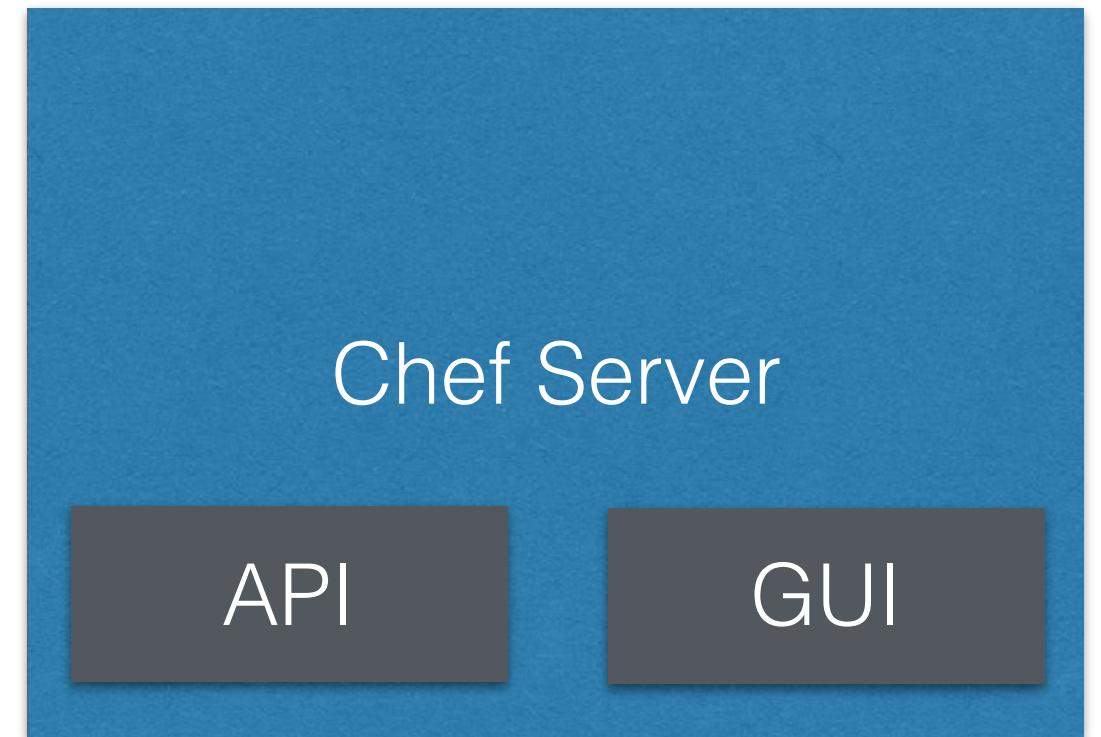
Chef-server

- The next step is to configure chef-server
- It will use chef to install chef-server on your machine

```
root@chef:~# chef-server-ctl reconfigure
...
Running handlers:
Running handlers complete
Chef Client finished, 414/481 resources updated in 154.052770683 seconds
Chef Server Reconfigured!
root@chef:~#
```

```
root@chef:~# chef-server-ctl install opscode-manage
...
root@chef:~# chef-server-ctl reconfigure
...
root@chef:~# opscode-manage-ctl reconfigure
...
root@chef:~#
```

192.168.0.253

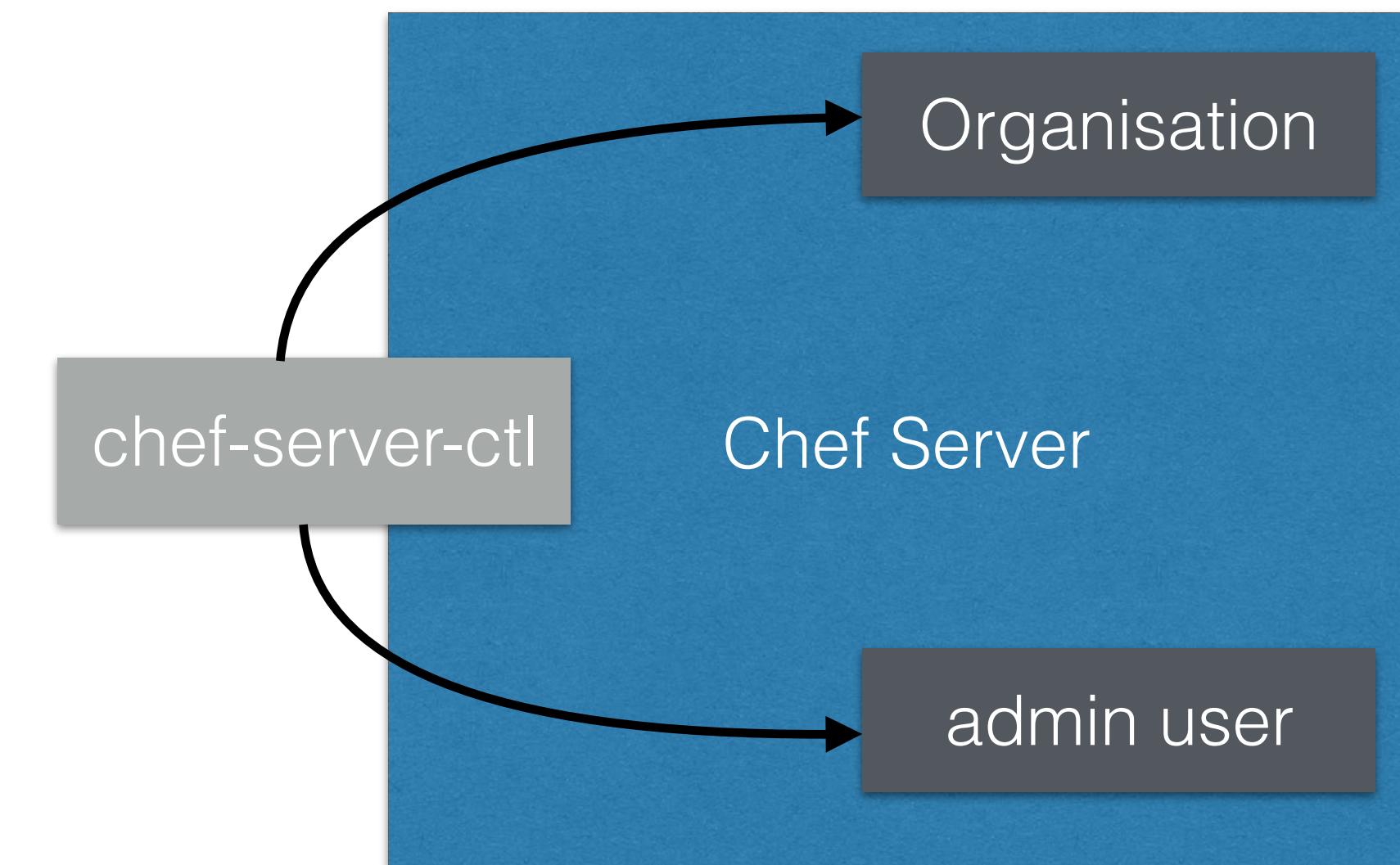


- Optionally you can install the GUI

Chef-server

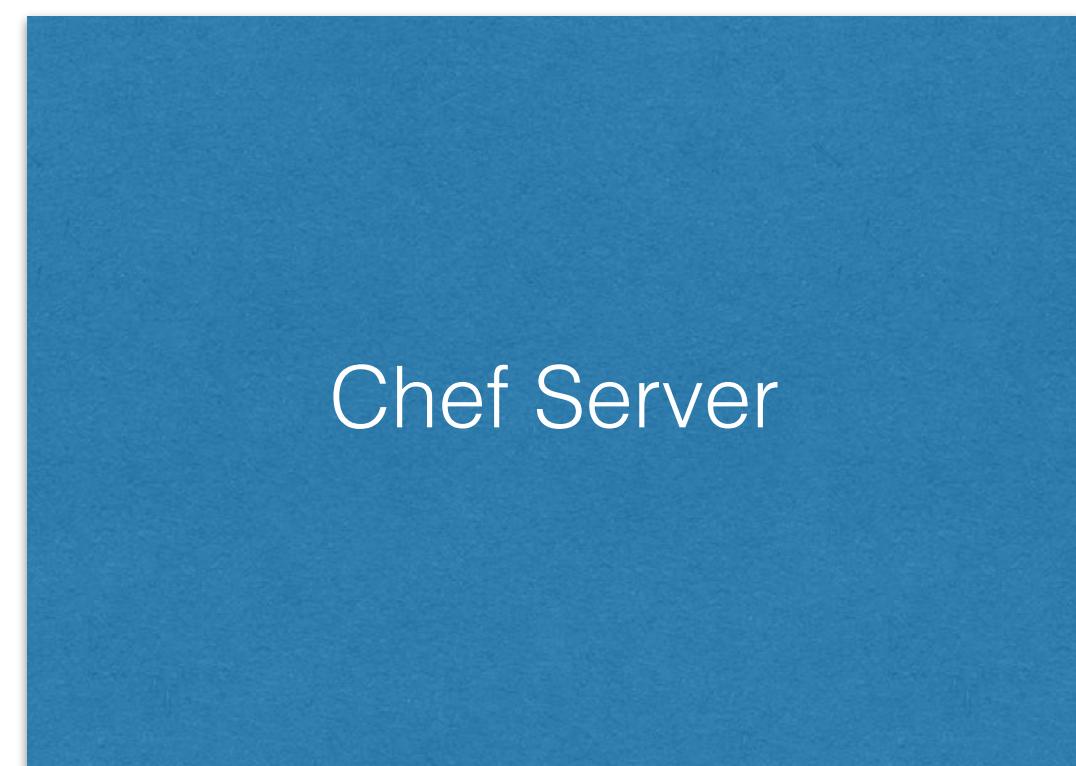
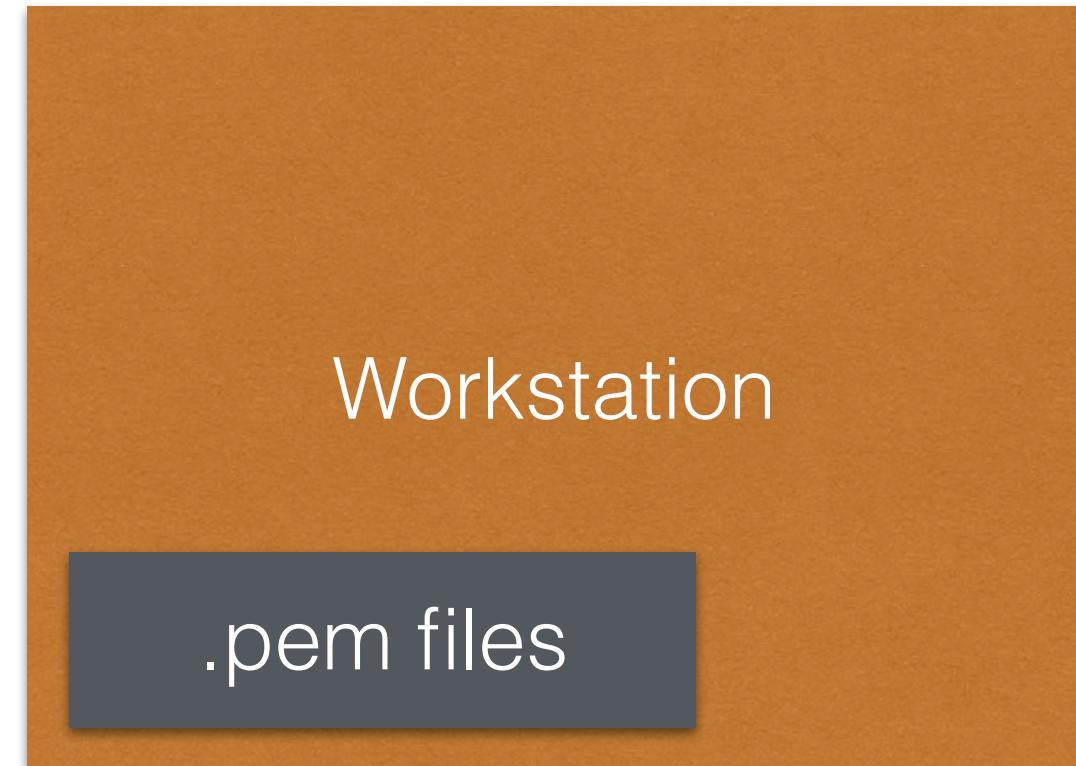
- Create an admin user and an organisation

```
root@chef:~# chef-server-ctl user-create admin admin admin@example.com LearnDevOps -f admin.pem  
root@chef:~# chef-server-ctl org-create learndevops "Learn DevOps Course" --association_user admin -f org.pem
```



Setting up the workstation

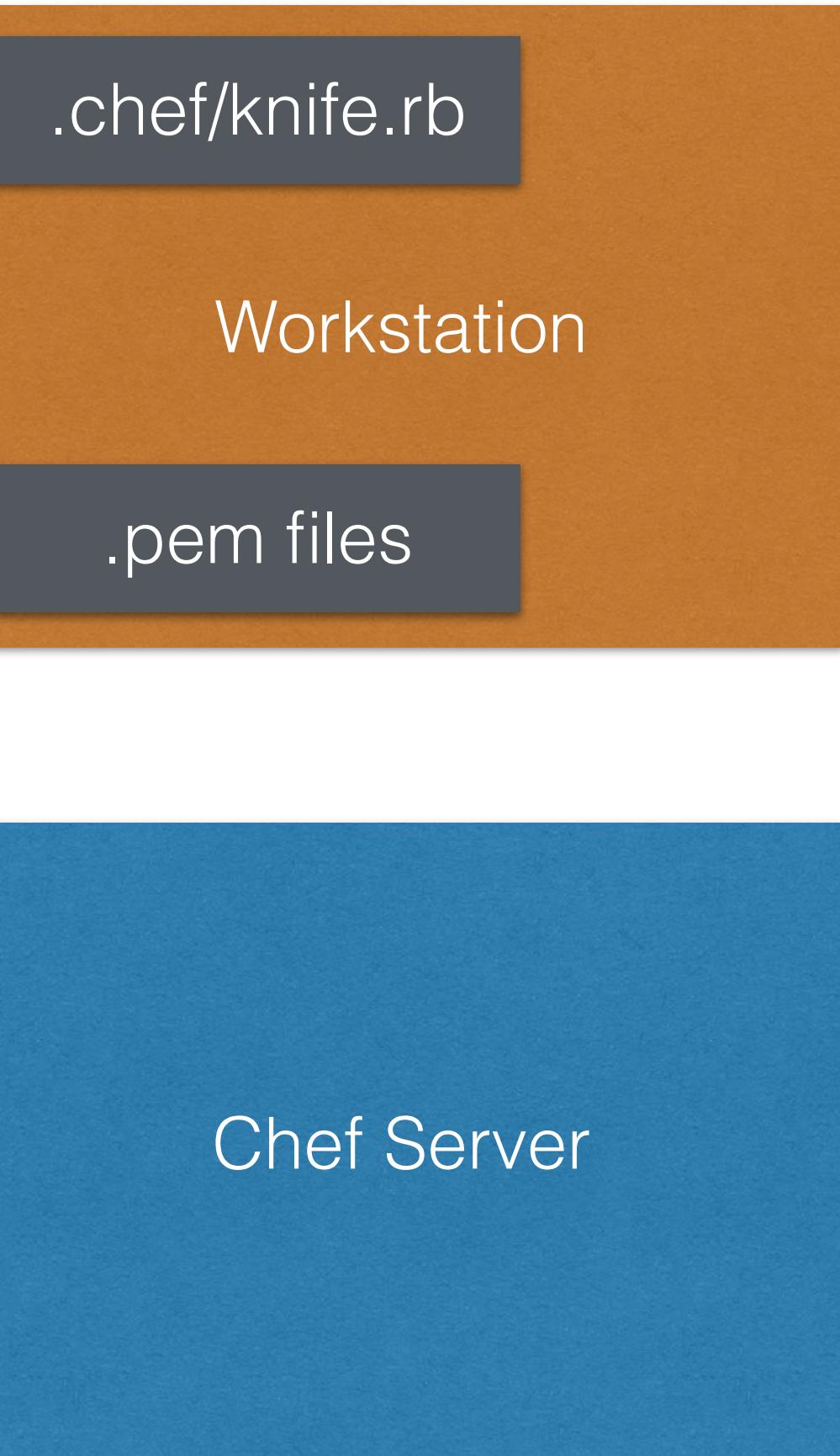
- With the 2 .pem files you can set up a workstation
 - You can use the workstation machine
 - You can use the same machine where the server is installed if you're low on resources
- You can download the knife.rb from the web GUI
 - <https://192.168.0.253> on your PC (Administration -> Generate Knife Config)
 - You can build the knife config manually



Setting up the workstation

- Building knife config

```
$ vagrant ssh workstation
vagrant@workstation:~$ mkdir .chef
vagrant@workstation:~$ cat << EOF > knife.rb
> # See https://docs.getchef.com/config_rb_knife.html for more information
>
> current_dir = File.dirname(__FILE__)
> log_level :info
> log_location STDOUT
> node_name "admin"
> client_key "/home/vagrant/admin.pem"
> validation_client_name "learndevops-validator"
> validation_key "/home/vagrant/org.pem"
> chef_server_url "https://chef.example.com/organizations/learndevops"
> cookbook_path ["~/home/vagrant/cookbooks"]
> EOF
vagrant@workstation:~$ cat knife.rb
```

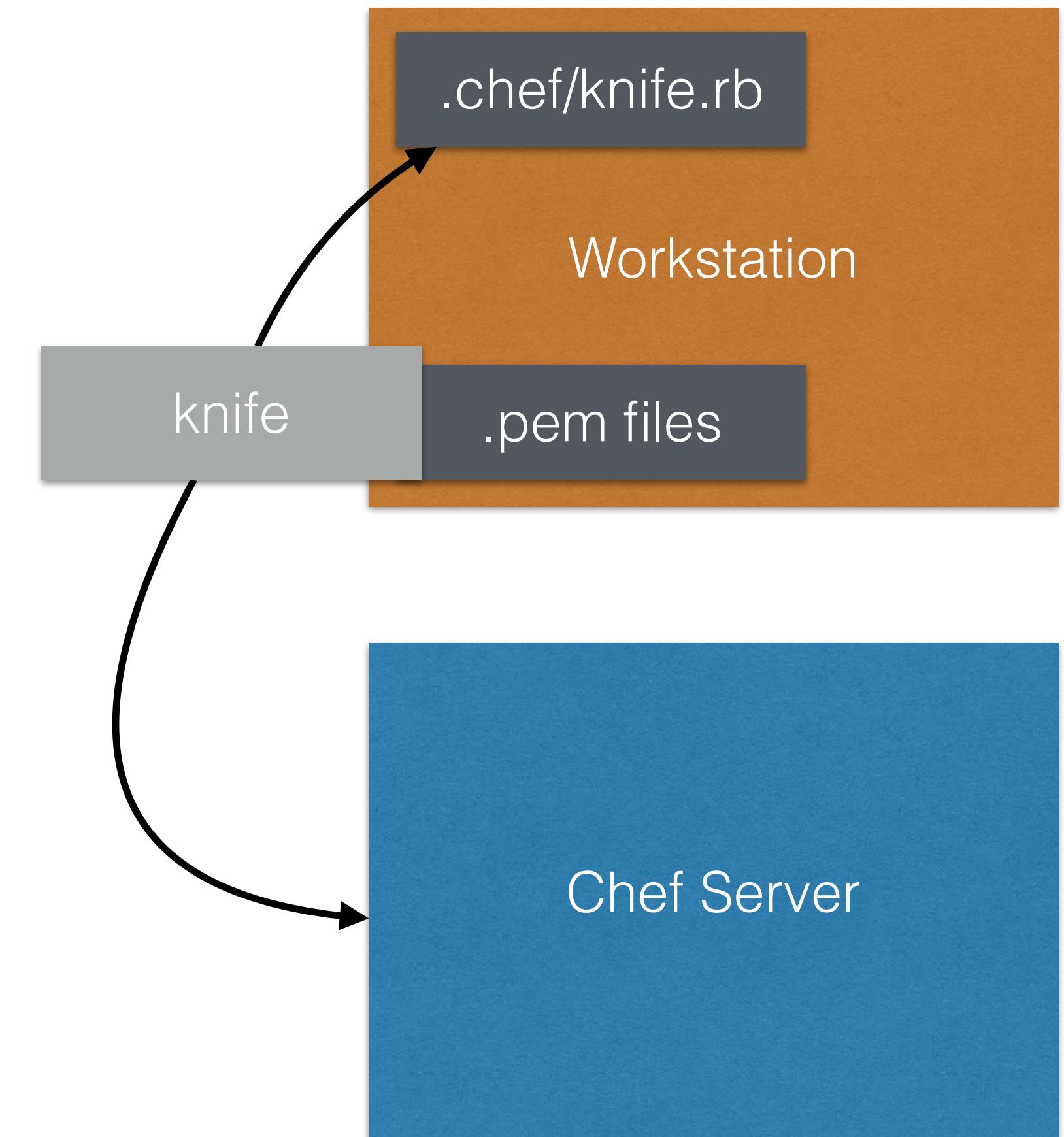


- copy paste the contents of admin.pem and org.pem to the same files on the workstation or use the shared vagrant folder to copy them across

Setting up the workstation

- test config

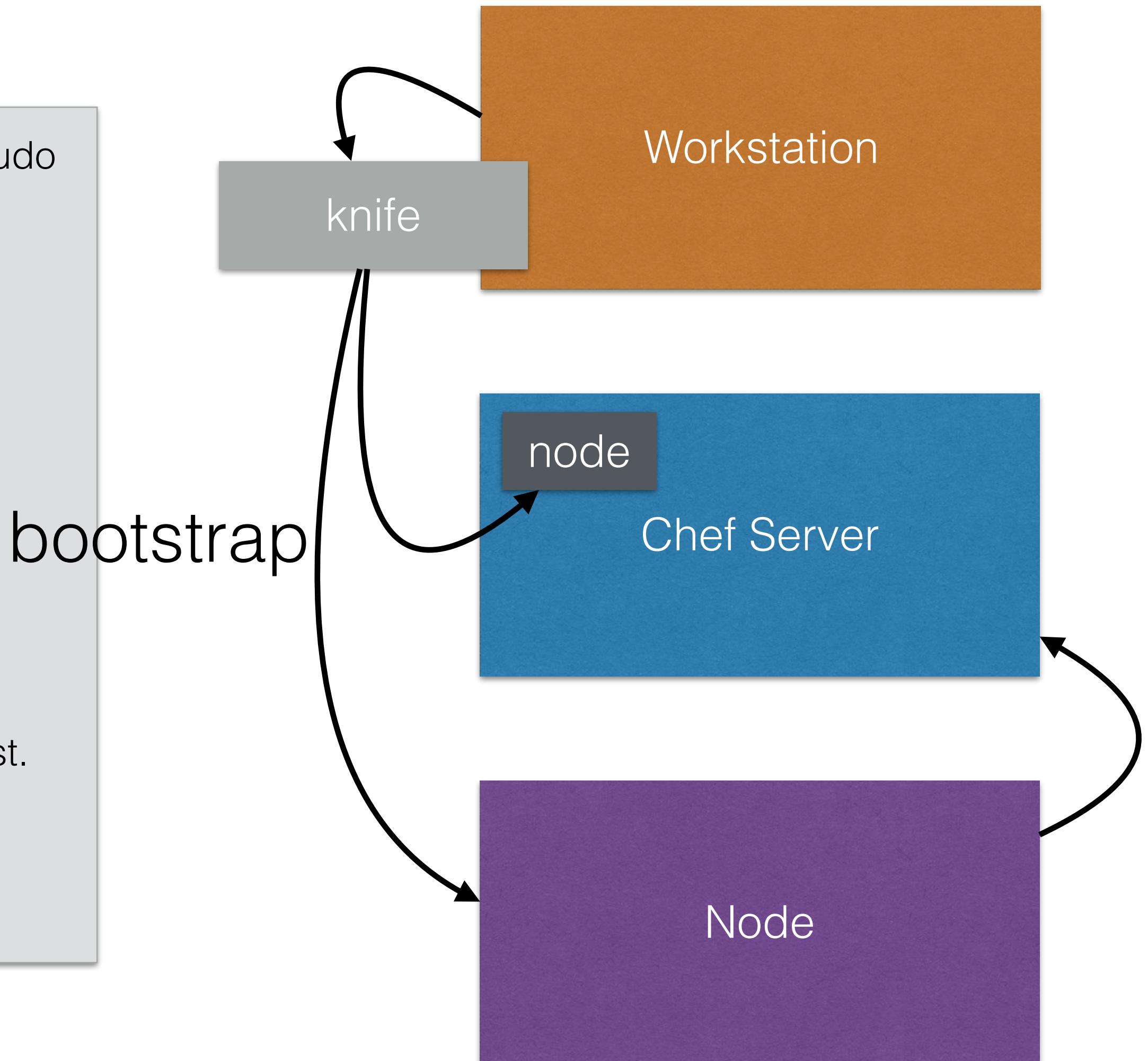
```
vagrant@workstation:~$ knife ssl fetch  
...  
vagrant@workstation:~$ knife client list  
learndevops-validator  
vagrant@workstation:~$
```



Bootstrapping

- Bootstrap a new node

```
vagrant@workstation:~$ knife bootstrap node.example.com -N node -x vagrant --sudo  
Creating new client for node  
Creating new node for node  
Connecting to node.in4it.io  
vagrant@node.example.com's password:  
node.in4it.io -----> Existing Chef installation detected  
node.in4it.io Starting the first Chef Client run...  
node.in4it.io Starting Chef Client, version 11.8.2  
node.in4it.io resolving cookbooks for run list: []  
node.in4it.io Synchronizing Cookbooks:  
node.in4it.io Compiling Cookbooks...  
node.in4it.io [2015-10-28T15:18:48+00:00] WARN: Node node has an empty run list.  
node.in4it.io Converging 0 resources  
node.in4it.io Chef Client finished, 0 resources updated  
vagrant@workstation:~$
```



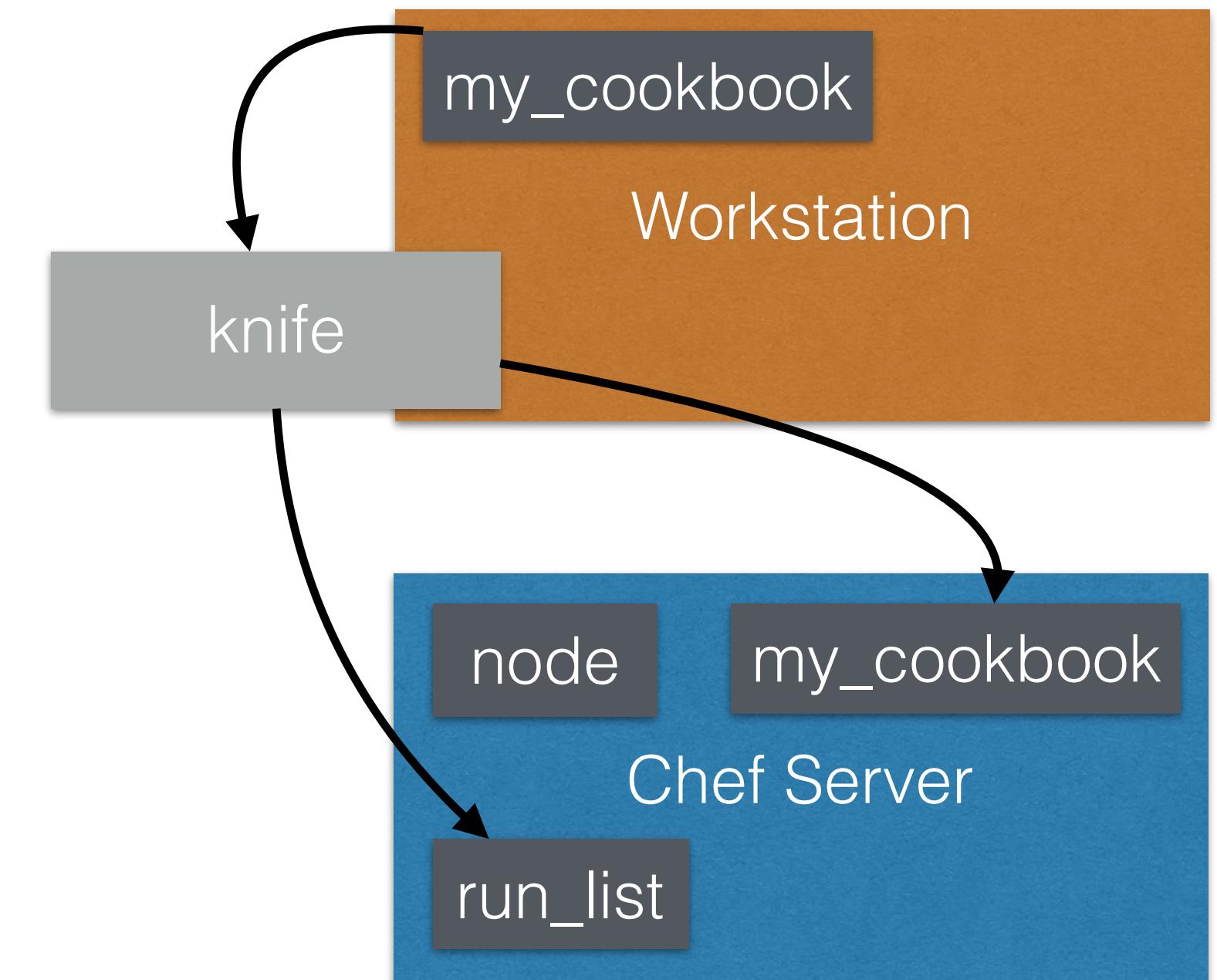
Running the cookbook

- Upload the cookbook we created earlier to the chef server

```
vagrant@workstation:~$ cd cookbooks  
vagrant@workstation:~/cookbooks$ knife cookbook site download apt  
vagrant@workstation:~/cookbooks$ tar -xzf apt-*.tar.gz  
vagrant@workstation:~/cookbooks$ knife cookbook upload apt  
vagrant@workstation:~/cookbooks$ knife cookbook upload my_cookbook
```

- add our new cookbook to node's run list

```
vagrant@workstation:~$ knife node run_list set node 'recipe[my_cookbook]'  
node:  
  run_list: recipe[my_cookbook]  
vagrant@workstation:~$
```



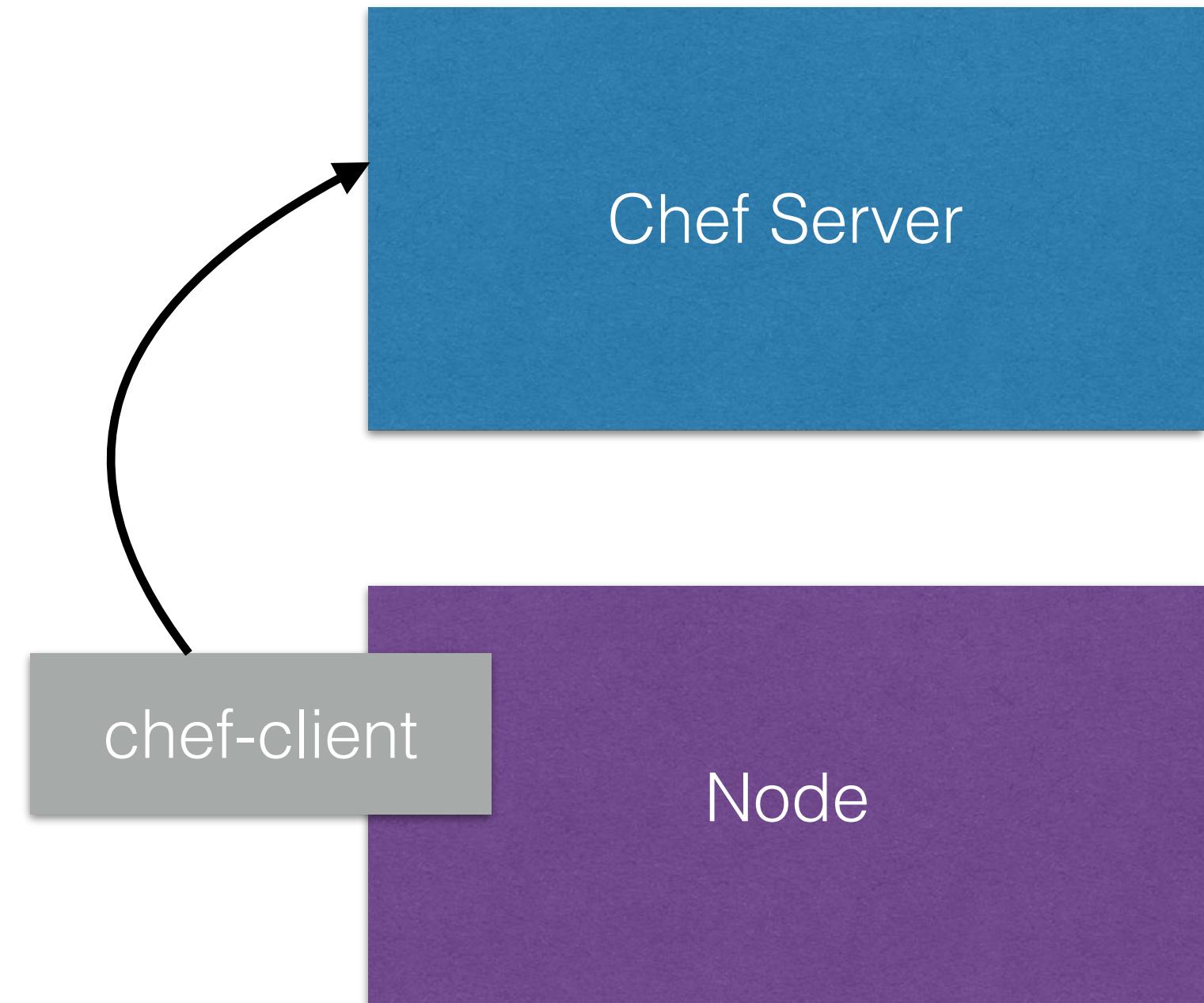
Running cookbook

- chef-client runs periodically (every 30 min by default), but let's force the run with knife to have the cookbook installed immediately

```
vagrant@workstation:~$ knife ssh 'id:*' 'sudo chef-client'  
...  
vagrant@workstation:~$
```

Or

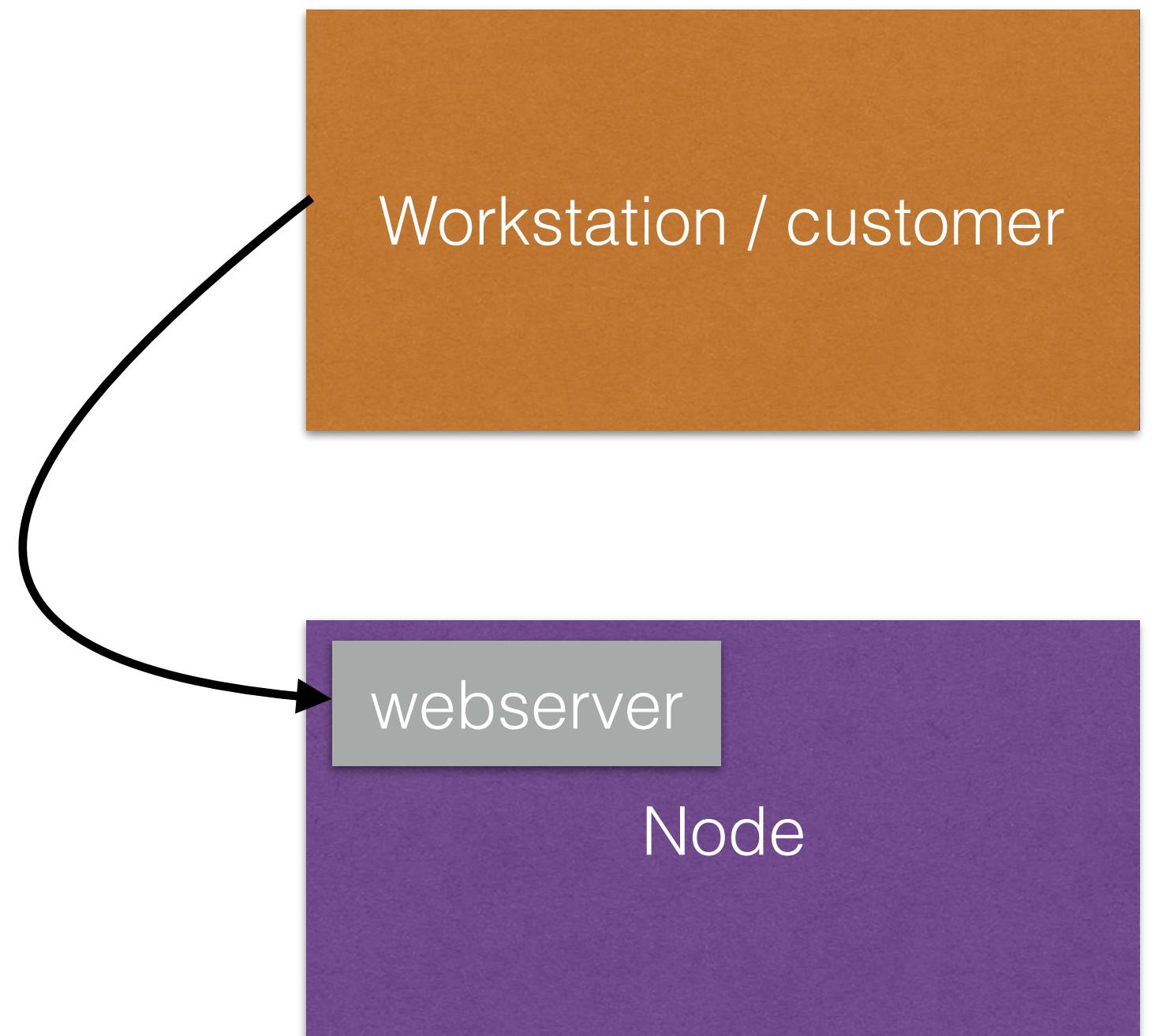
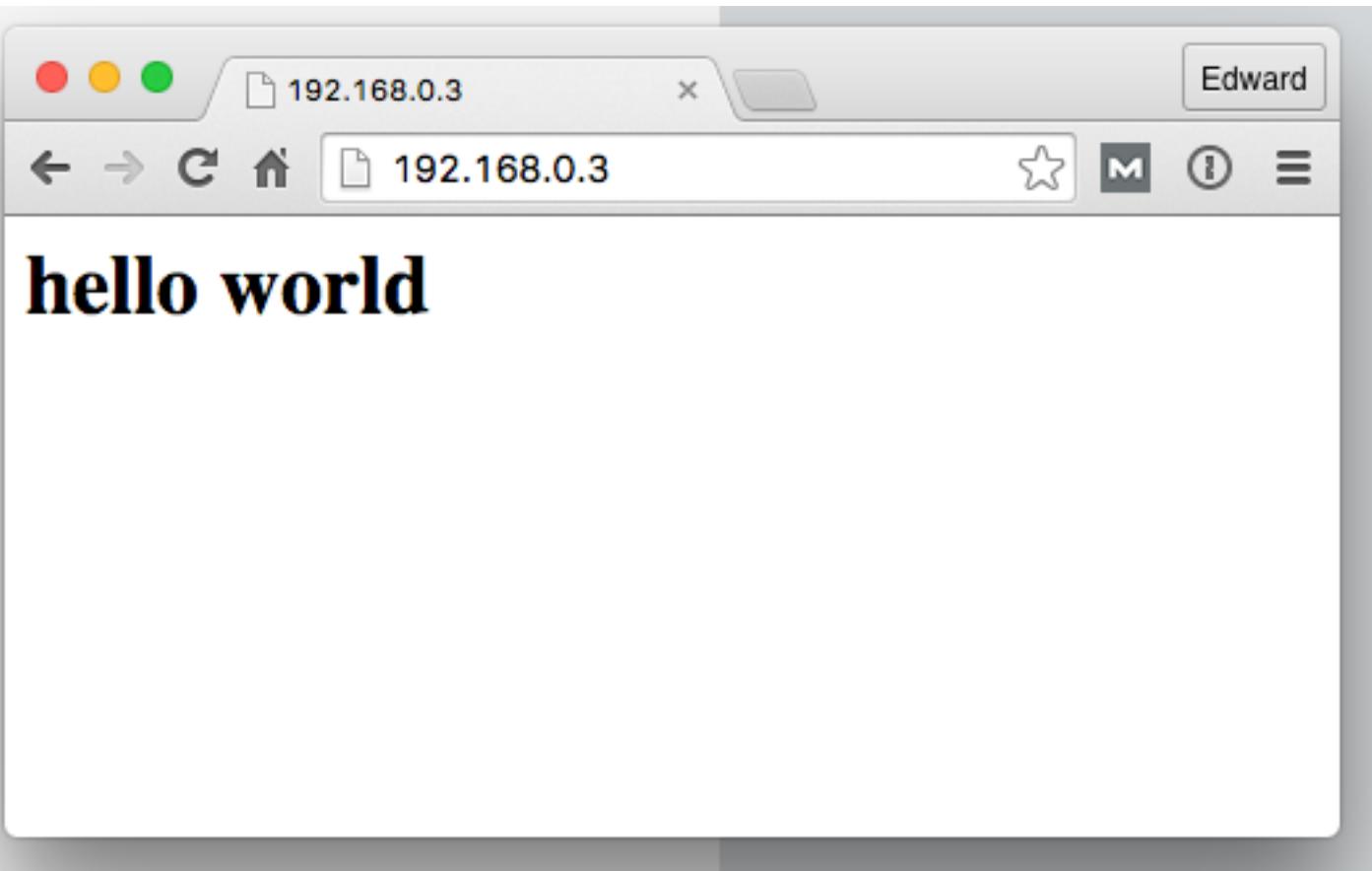
```
vagrant@chef:~$ ssh node.example.com  
vagrant@node:~$ sudo chef-client  
...  
vagrant@node:~$
```



Running cookbook

- Let's test

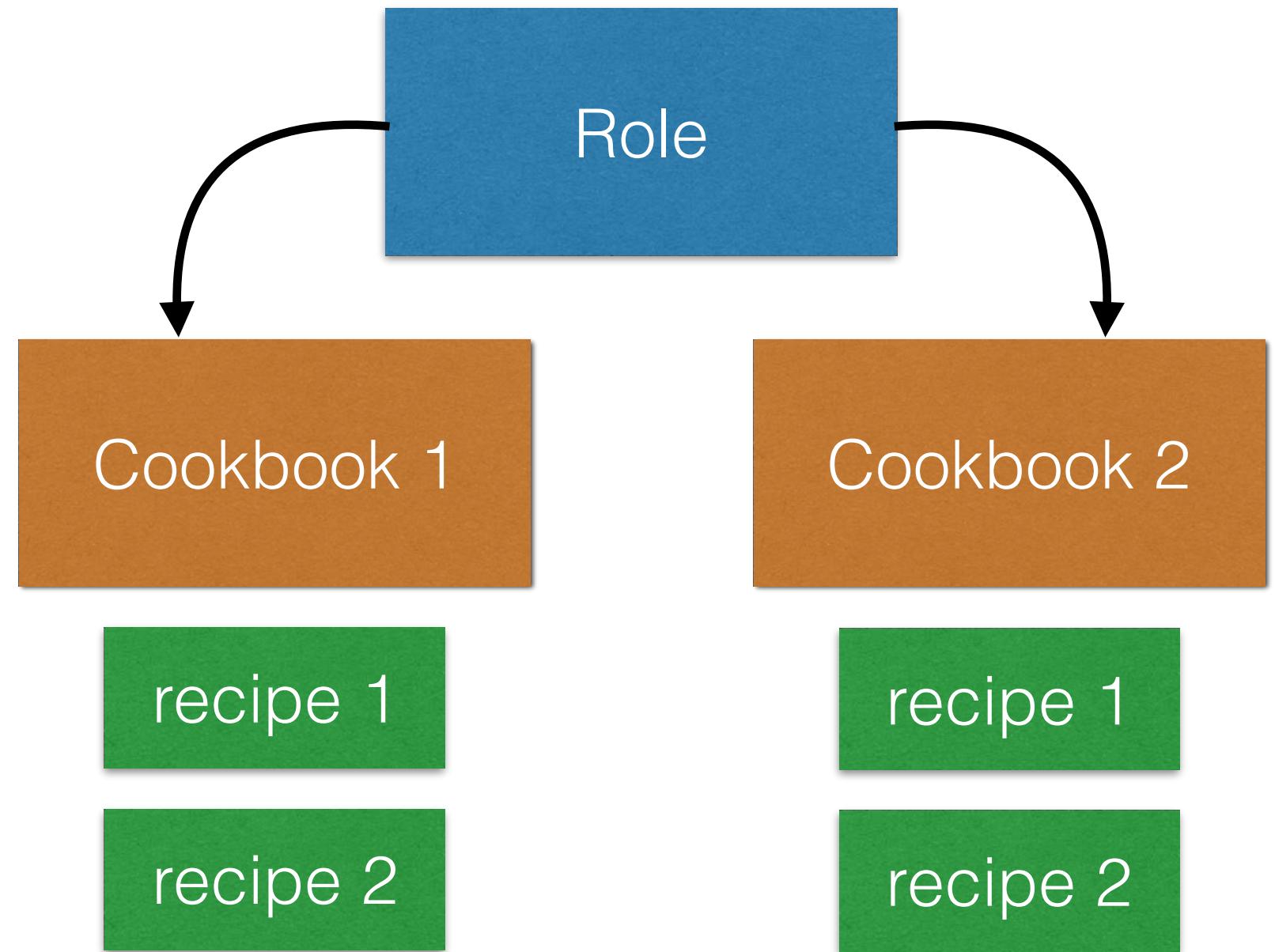
```
vagrant@workstation:~/cookbooks$ curl node
<html>
  <body>
    <h1>hello world</h1>
  </body>
</html>
vagrant@workstation:~/cookbooks$
```



Using roles

- It's better to use roles
- We need to create a file webserver-role.json

```
{  
  "name": "webserver",  
  "default_attributes": {  
  },  
  "json_class": "Chef::Role",  
  "run_list": ["recipe[my_cookbook]"],  
  "description": "webserver role that installs my_cookbook",  
  "chef_type": "role",  
  "override_attributes": {  
  }  
}
```

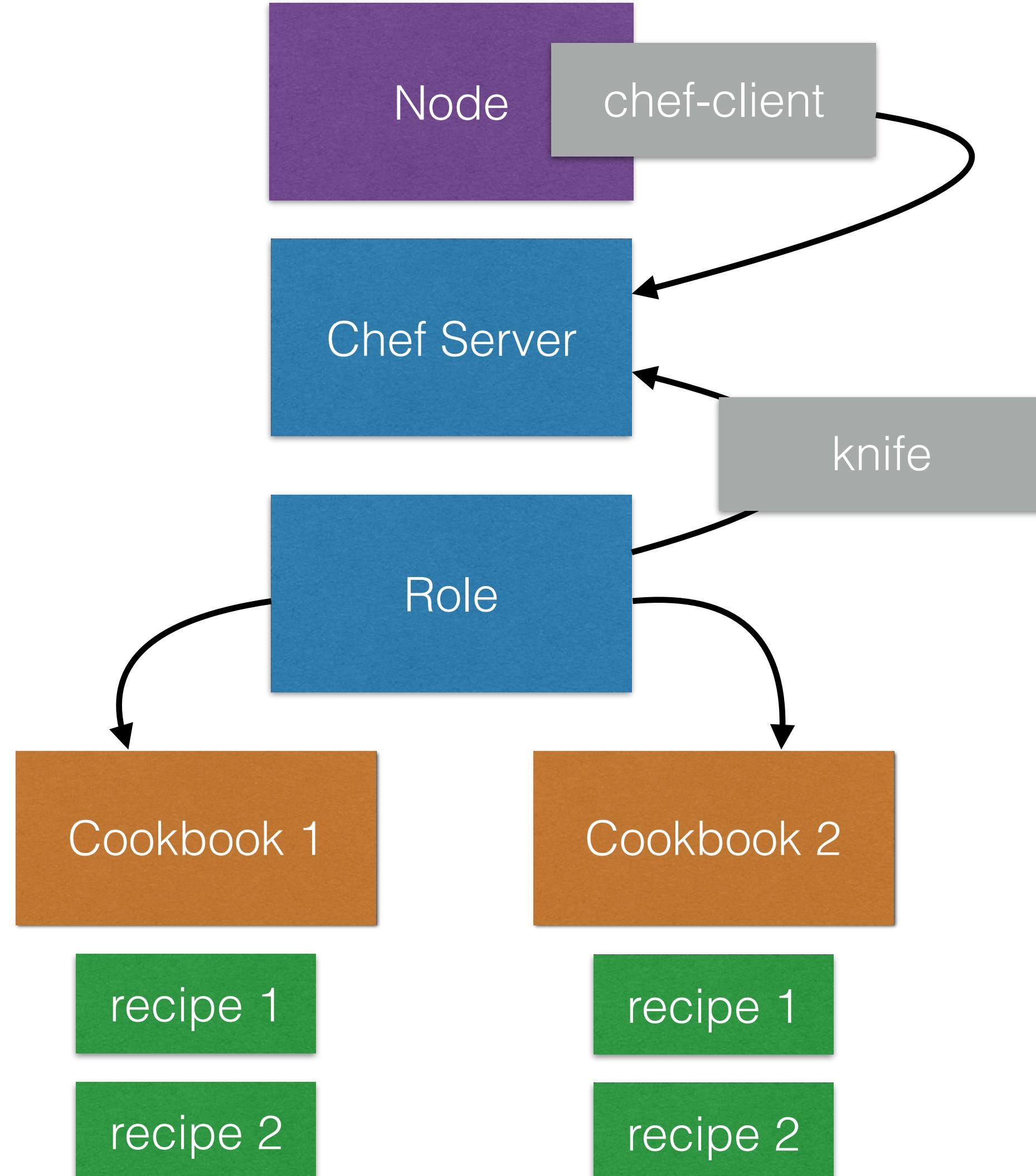


Using roles

- Upload the webserver-role.json to the chef server:

```
vagrant@workstation:~$ knife role from file webserver-role.json
Updated Role webserver!
vagrant@workstation:~$
```

```
vagrant@chef:~$ knife node run_list set node 'role[webserver]'
node:
  run_list: role[webserver]
vagrant@chef:~$
```



Separating template

- It's better to separate template files
- Contents of nginx.conf.erb

```
user <%= node[:nginx][:user] %>;
worker_processes <%= node[:nginx][:worker_processes] %>;
pid <%= node[:nginx][:pid] %>;
events {
    worker_connections <%= node[:nginx][:worker_connections] %> ;
}
http {
    sendfile on;
    tcp_nopush on;
    tcp_nodelay on;
    keepalive_timeout 65;
    types_hash_max_size 2048;
    include /etc/nginx/mime.types;
    default_type application/octet-stream;
    access_log /var/log/nginx/access.log;
    error_log /var/log/nginx/error.log;
    gzip on;
    gzip_disable "msie6";
    include /etc/nginx/conf.d/*.conf;
    include /etc/nginx/sites-enabled/*;
}
```



Separating template

- Separate the variables as well
- Contents of attributes/default.rb

```
default[:nginx][:user] = "www-data"
default[:nginx][:worker_processes] = "2"
default[:nginx][:pid] = "/run/nginx.pid"
default[:nginx][:worker_connections] = "768"
```

my_cookbook

metadata

readme

recipes

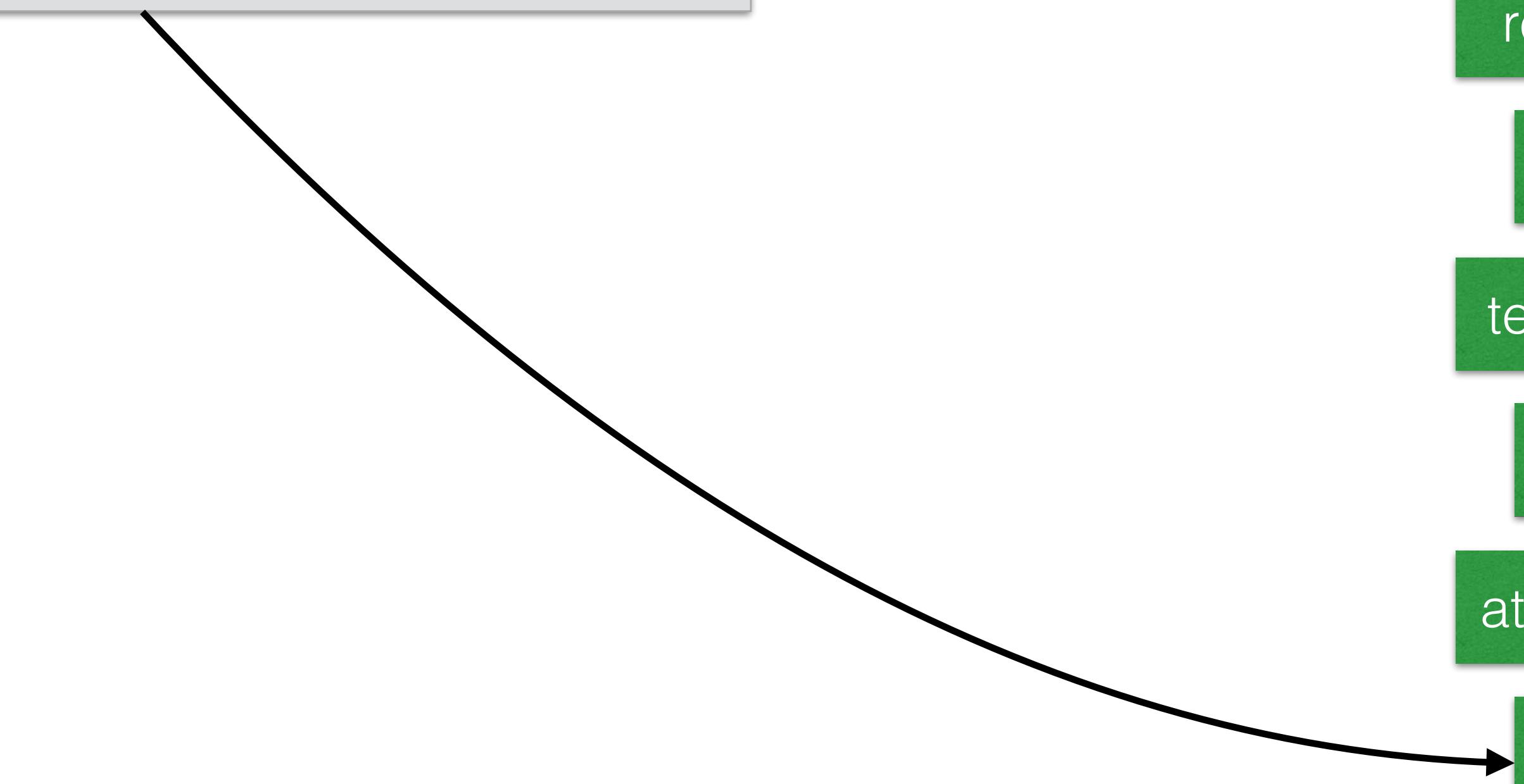
default.rb

template

nginx.conf.erb

attributes

default.rb



Separating template

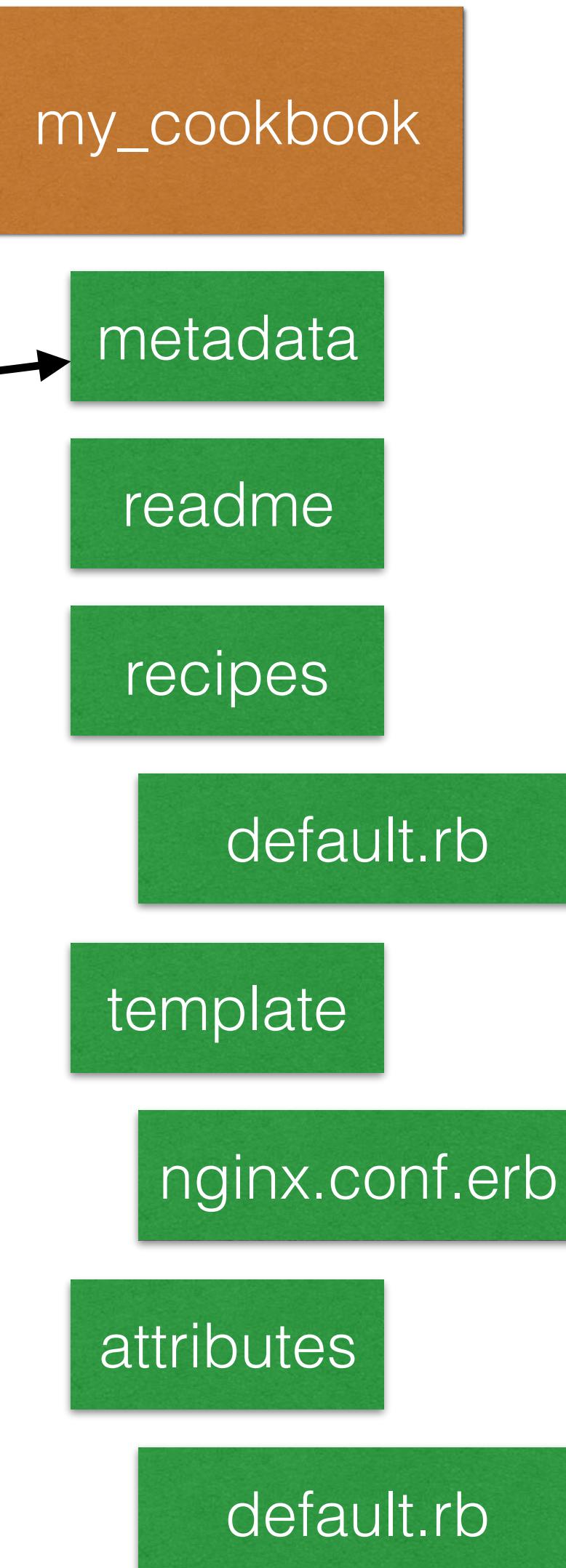
- Increase version number:

```
name      'my_cookbook'  
maintainer  'YOUR_COMPANY_NAME'  
maintainer_email 'YOUR_EMAIL'  
license     'All rights reserved'  
description  'Installs/Configures test'  
long_description IO.read(File.join(File.dirname(__FILE__), 'README.md'))  
version    '0.1.4'  
depends     'apt'
```

- Upload new version to the chef-server:

```
vagrant@workstation:~/cookbooks/my_cookbook$ knife cookbook upload my_cookbook  
Uploading my_cookbook [0.1.4]  
Uploaded 1 cookbook.  
vagrant@workstation:~/cookbooks/my_cookbook$
```

- By default it will be picked up in 30 min, to run manually, run “sudo chef-client”



Chef-server demo

AWS OpsWorks

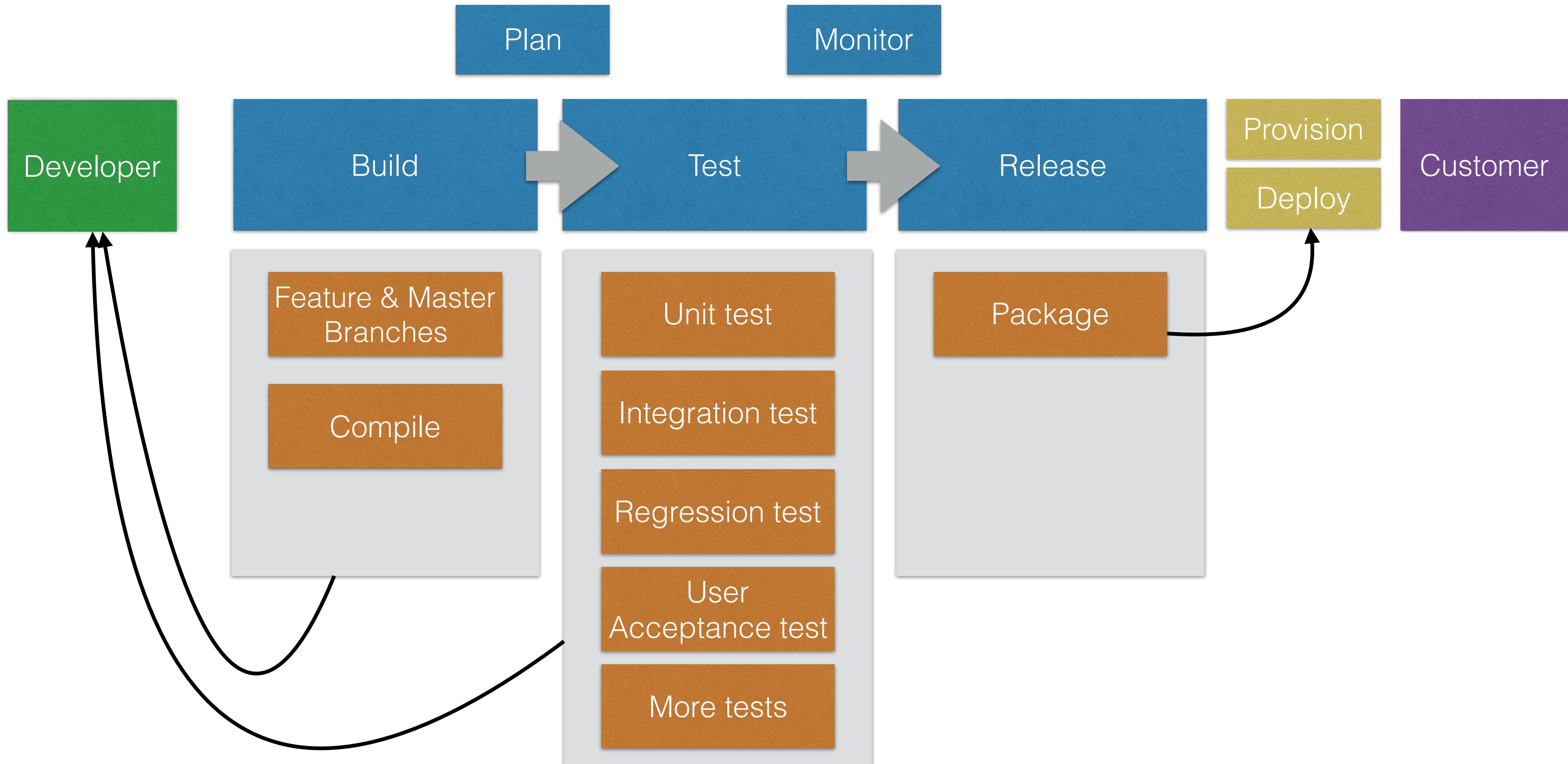
Based on Chef

What is OpsWorks

- AWS's automation and configuration management platform based on Chef
- Chef cookbooks can be used in OpsWorks
- Provisions EC2 machines
- Can provision on-premise machines for an extra fee

Step 3 - Continuous Integration

Continuous Integration



Continuous Integration tools

- Jenkins
- Bamboo (hosted or install - Atlassian)
- TeamCity (JetBrains)
- CircleCI
- Codeship's ParallelCI
- Travis CI (can build your open source github project for free)
- AWS CodePipeLine

Jenkins

- Open source continuous integration tool written in Java.
- Project was forked from another project called Hudson, after a dispute with Oracle
- There are a lot of plugins available for Jenkins
- There is easier to use CI software available, but Jenkins is open source, free and still popular

Setting up Jenkins

- New Vagrantfile
- Create a new project folder called “ci” and create Vagrantfile

```
Vagrant.configure(2) do |config|
  config.vm.define "jenkins" do |jenkins|
    jenkins.vm.box = "ubuntu/trusty64"
    jenkins.vm.network "private_network", ip: "192.168.0.252"
    jenkins.vm.hostname = "jenkins"
    jenkins.vm.provider "virtualbox" do |v|
      v.memory = 4096
      v.cpus = 2
    end
  end
end
```

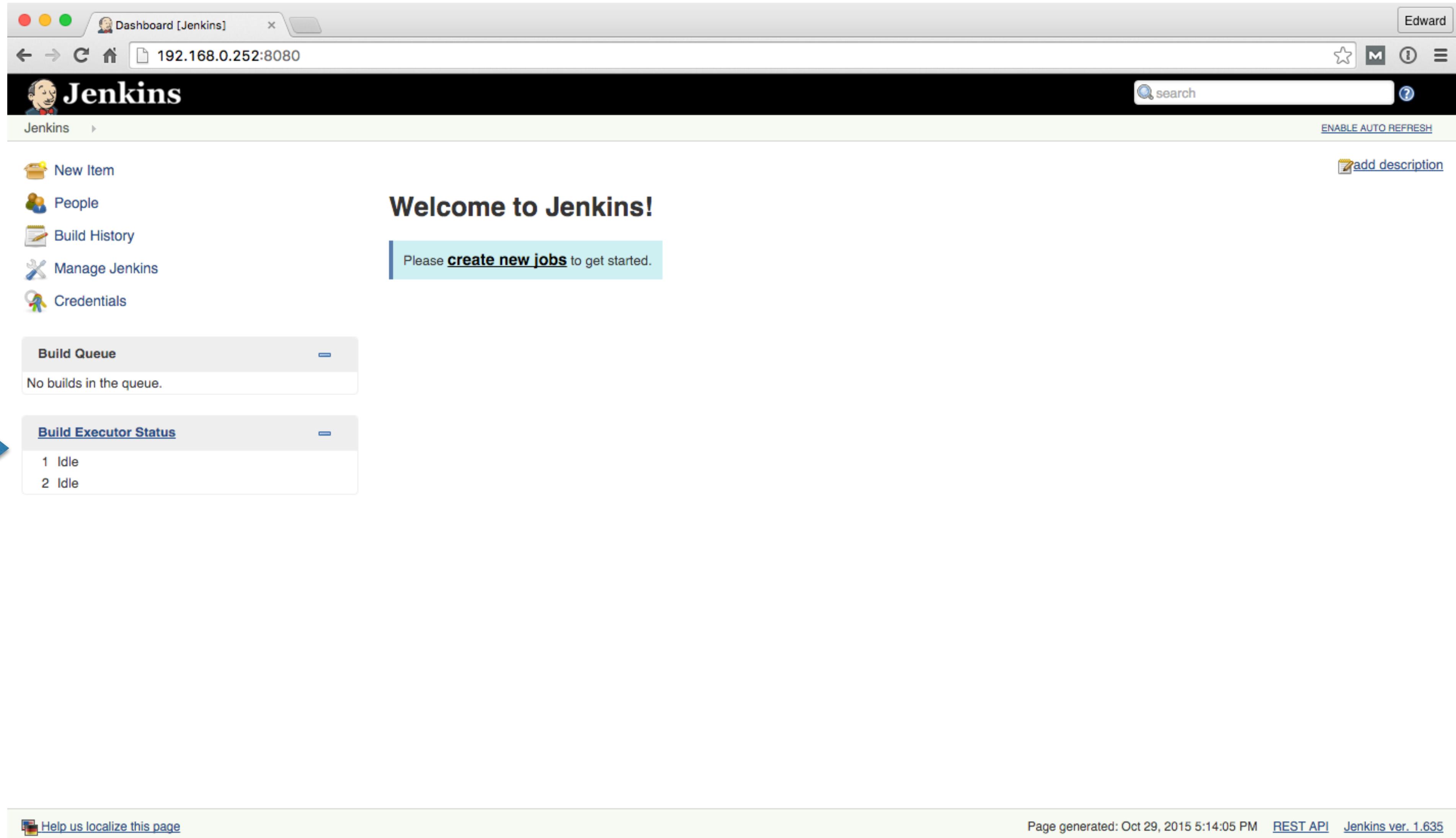
```
$ vagrant up jenkins
$ vagrant ssh jenkins
```

Setting up Jenkins

- Install jenkins
 - Using chef / ansible
 - Using manual configuration

```
vagrant@jenkins:~$ wget -q -O - http://pkg.jenkins-ci.org/debian/jenkins-ci.org.key | sudo apt-key add -
vagrant@jenkins:~$ sudo -s
root@jenkins:~# echo "deb http://pkg.jenkins-ci.org/debian binary/" >> /etc/apt/sources.list
root@jenkins:~# apt-get update
...
root@jenkins:~# apt-get install jenkins
...
root@jenkins:~# service jenkins start
root@jenkins:~#
```

Setting up Jenkins



A screenshot of a web browser displaying the Jenkins dashboard at 192.168.0.252:8080. The page title is "Dashboard [Jenkins]". The main content area features a "Welcome to Jenkins!" message with a call to action: "Please [create new jobs](#) to get started." On the left sidebar, there are links for "New Item", "People", "Build History", "Manage Jenkins", and "Credentials". Below these are two expandable sections: "Build Queue" (which shows "No builds in the queue.") and "Build Executor Status" (which shows "1 Idle" and "2 Idle"). A large blue arrow points from the left towards the "Build Executor Status" section. At the bottom of the page, there is a link to "Help us localize this page" and footer information including the page generation time ("Page generated: Oct 29, 2015 5:14:05 PM"), a "REST API" link, and the Jenkins version ("Jenkins ver. 1.635").

Setting up the app

- Scala/Java app
 - You don't need to able to understand Scala or java
 - When doing DevOps you will be exposed to many different languages
- Continuous Integration is little bit different for every language
- Ops needs to work together with Devs to get CI right

Play framework

- Open Source Web Application Framework written in scala and java
- Follow model-view-controller (MVC) architectural pattern
- Stateless, fully RESTful
- API
- Async I/O
- Modular
- Native scala support

Setting up the app

- Step 1: install java 8

```
vagrant@jenkins:~$ sudo add-apt-repository ppa:webupd8team/java -y  
vagrant@jenkins:~$ sudo apt-get update  
vagrant@jenkins:~$ sudo apt-get install oracle-java8-installer
```

- install sbt

```
vagrant@jenkins:~$ echo "deb https://dl.bintray.com/sbt/debian /" | sudo tee -a /etc/apt/sources.list.d/sbt.list  
vagrant@jenkins:~$ sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 642AC823  
vagrant@jenkins:~$ sudo apt-get update  
vagrant@jenkins:~$ sudo apt-get install sbt
```

Setting up the app

- Step 2: install play! framework
- Go to <https://www.playframework.com/download> and get latest version or use wget:

```
vagrant@jenkins:~$ wget https://downloads.typesafe.com/typesafe-activator/1.3.6/typesafe-activator-1.3.6-minimal.zip
```

- Install activator

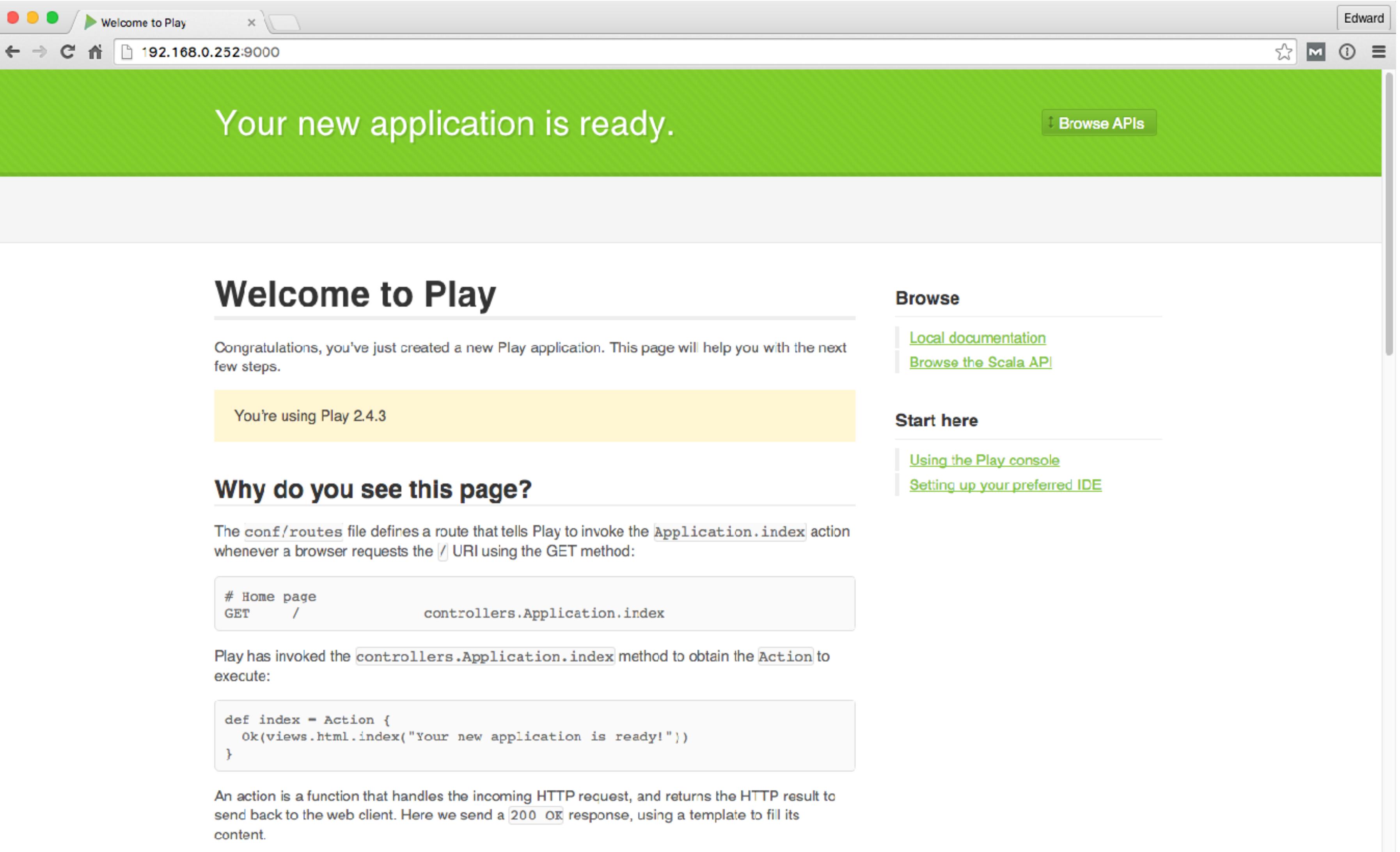
```
vagrant@jenkins:~$ unzip typesafe-activator-*zip  
vagrant@jenkins:~$ mv activator-1.3.6-minimal/ bin  
vagrant@jenkins:~$ . .profile
```

Running the app

- Step 3: Create the example app

```
vagrant@jenkins:~$ activator new example-app play-java  
vagrant@jenkins:~$ cd example-app  
vagrant@jenkins:~/example-app$ ./activator run
```

Setting up the app



The screenshot shows a web browser window titled "Welcome to Play" with the URL "192.168.0.252:9000". The main content area has a green header bar with the text "Your new application is ready." and a "Browse APIs" button. Below this, the "Welcome to Play" page is displayed. It includes a yellow callout box stating "You're using Play 2.4.3". The "Why do you see this page?" section explains that the routes file defines a route to invoke the Application.index action. It shows the corresponding code snippet from the routes file: "# Home page GET / controllers.Application.index". The "Action" part of the explanation is expanded to show the Java code: "def index = Action { Ok(views.html.index("Your new application is ready!")) }". A note at the bottom states that an action handles the incoming HTTP request and returns the HTTP result to send back to the client.

Your new application is ready.

Edward

Welcome to Play

Congratulations, you've just created a new Play application. This page will help you with the next few steps.

You're using Play 2.4.3

Why do you see this page?

The `conf/routes` file defines a route that tells Play to invoke the `Application.index` action whenever a browser requests the `/` URI using the GET method:

```
# Home page
GET / controllers.Application.index
```

Play has invoked the `controllers.Application.index` method to obtain the `Action` to execute:

```
def index = Action {
  Ok(views.html.index("Your new application is ready!"))
}
```

An action is a function that handles the incoming HTTP request, and returns the HTTP result to send back to the web client. Here we send a `200 OK` response, using a template to fill its content.

Browse

[Local documentation](#)
[Browse the Scala API](#)

Start here

[Using the Play console](#)
[Setting up your preferred IDE](#)

Building and Packaging Apps

- Depending on the programming language, a specific building and packaging tool will be used
 - For Java projects, you will often see ant, maven or gradle
 - For Scala projects, sbt is often used
- Other languages that don't need to be compiled still have tools available that you can use for dependency management and to run unit tests
 - Grunt and npm are popular for NodeJS
 - Composer for dependency management in PHP
 - pip and easy_install for Python

Maven vs sbt

- When building Java/Scala, you will often come across maven and sbt
- Both can be used to build Java and Scala projects
- You see maven being used for a lot of open source projects
- sbt is commonly used across Scala
- They have the same purpose, both of them have pros and cons

Apache Maven

- Configuration is done using a pom.xml
- Supports plugins (there is for instance a plugin to build rpm's)
- Maven has dependency management, it will download dependencies, called artifacts, for you
- Integrates well with popular IDEs
- Maven will expect a standard project structure
 - src/main/java for java project source code
 - src/test/java for java unit tests

Example pom file

```
1. <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2.   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
3.   <modelVersion>4.0.0</modelVersion>
4.
5.   <groupId>com.mycompany.app</groupId>
6.   <artifactId>my-app</artifactId>
7.   <version>1.0-SNAPSHOT</version>
8.   <packaging>jar</packaging>
9.
10.  <name>Maven Quick Start Archetype</name>
11.  <url>http://maven.apache.org</url>
12.
13.  <dependencies>
14.    <dependency>
15.      <groupId>junit</groupId>
16.      <artifactId>junit</artifactId>
17.      <version>4.8.2</version>
18.      <scope>test</scope>
19.    </dependency>
20.  </dependencies>
21. </project>
```

sbt

- Less effort than Maven for simple projects
- Integrates most tasks you will need, but has less plugins available than Maven
- Supports incremental compilation (gives you performance gains)
- Interactive console allows modifying build settings on the fly
- Newer than maven, integration with IDEs/CIs is not always at the same maturity level

build.sbt example

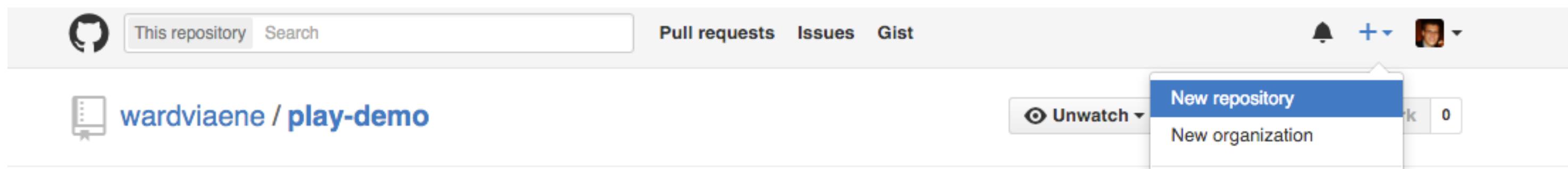
27 lines (16 sloc) | 539 Bytes

Raw Blame History

```
1 name := """example-app"""
2
3 version := "1.0-SNAPSHOT"
4
5 lazy val root = (project in file(".")).enablePlugins(PlayJava)
6
7 scalaVersion := "2.11.6"
8
9 libraryDependencies ++= Seq(
10   javaJdbc,
11   cache,
12   javaws
13 )
14
15 // Play provides two styles of routers, one expects its actions to be injected, the
16 // other, legacy style, accesses its actions statically.
17 routesGenerator := InjectedRoutesGenerator
18
19 enablePlugins(DebianPlugin)
20
21 maintainer := "Edward Viaene <ward@in4it.io>"
22
23 packageSummary := "My custom package"
24
25 packageDescription := "Package"
26
```

Integrate with version control

- Let's enable version control for our newly created codebase
- Log in to github.com (or any other repository server)
 - create a new public repository



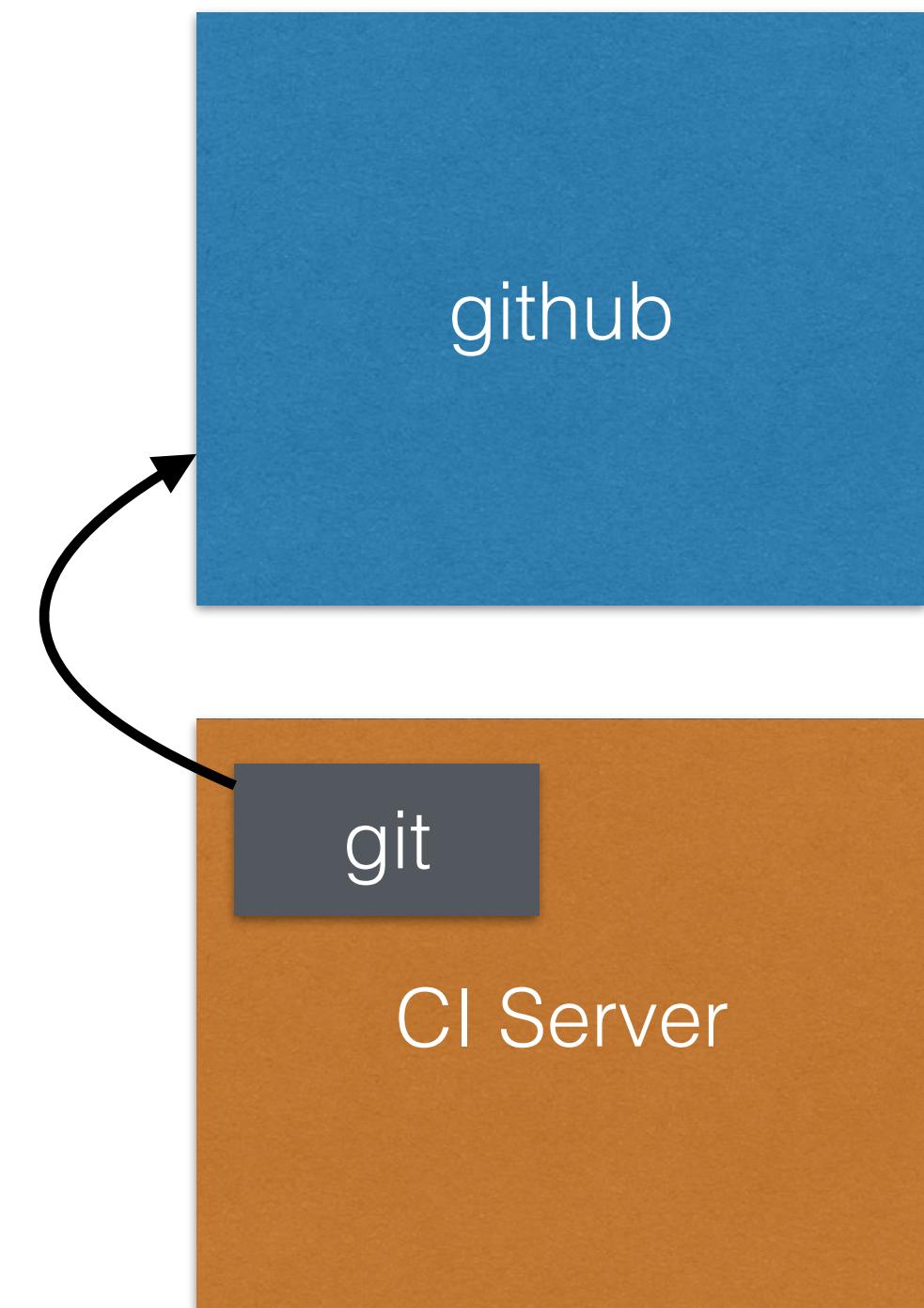
- Give it a name, like play demo:

A screenshot of the 'Create a new repository' form on GitHub. The form has a title 'Create a new repository' and a subtitle 'A repository contains all the files for your project, including the revision history.' Below the subtitle, there are two input fields: 'Owner' and 'Repository name'. The 'Owner' field shows 'wardviaene'. The 'Repository name' field shows 'play-demo' with a red warning icon next to it. There is also a small red warning icon at the bottom left of the form area.

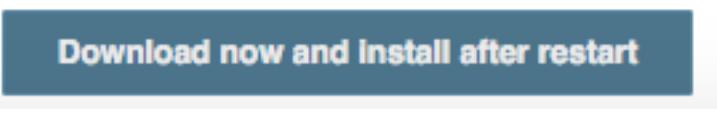
Integrate with version control

- Initialize, add, commit, and push the code to the newly created repository:
 - To quit the app if it's still running, press ctrl+d

```
vagrant@jenkins:~/example-app$ sudo apt-get install git  
...  
vagrant@jenkins:~/example-app$ git init  
vagrant@jenkins:~/example-app$ git config --global user.email "your@email"  
vagrant@jenkins:~/example-app$ git config --global user.name "your name"  
vagrant@jenkins:~/example-app$ git add .  
vagrant@jenkins:~/example-app$ git commit -am "initial commit"  
vagrant@jenkins:~/example-app$ git remote add origin https://github.com/YOURUSERNAME/play-demo.git  
vagrant@jenkins:~/example-app$ git push -u origin master
```



Setting up the app in jenkins

- First, you will need some plugins:
 - Git plugin
 - sbt plugin
- Go to Jenkins: <http://192.168.0.252:8080/>
- Go to manage jenkins:  [Manage Jenkins](#)
- Then click on Manage Plugins:  [Manage Plugins](#)
Add, remove, disable or enable plugins that can extend the functionality of Jenkins. **(updates available)**
- Select git plugin and sbt plugin and click on: 

Configure jvm and sbt

- Configuration changes
 - Click on Manage Jenkins
 - Click on Configure System
 - Configure JVM:
 - JAVA_HOME: /usr/lib/jvm/java-8-oracle/



JDK

JDK installations	
	Name <input type="text" value="java-8"/>
	JAVA_HOME <input type="text" value="/usr/lib/jvm/java-8-oracle/"/>
<input type="checkbox"/> Install automatically ?	

Configure jvm and sbt

- Configure sbt:
 - sbt launch jar: /usr/share/sbt-launcher-packaging/bin/sbt-launch.jar
 - Make sure you add -Xmx512M in the launch arguments

Sbt

Sbt installations						
<table border="1"><tr><td>Sbt</td><td><input type="text" value="sbt"/></td></tr><tr><td>sbt launch jar</td><td><input type="text" value="/usr/share/sbt-launcher-packaging/bin/sbt-launch.jar"/></td></tr><tr><td>sbt launch arguments</td><td><input type="text" value="-Xmx512M"/></td></tr></table> <p><input type="checkbox"/> Install automatically ?</p> <p>Delete Sbt</p> <p>Add Sbt</p> <p>List of Sbt installations on this system</p>	Sbt	<input type="text" value="sbt"/>	sbt launch jar	<input type="text" value="/usr/share/sbt-launcher-packaging/bin/sbt-launch.jar"/>	sbt launch arguments	<input type="text" value="-Xmx512M"/>
Sbt	<input type="text" value="sbt"/>					
sbt launch jar	<input type="text" value="/usr/share/sbt-launcher-packaging/bin/sbt-launch.jar"/>					
sbt launch arguments	<input type="text" value="-Xmx512M"/>					

Setting up jenkins

- Now it's time to create a new job:
- Create an example-app project:

Welcome to Jenkins!

Please [create new jobs](#) to get started.

Item name

Freestyle project
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

Maven project
Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.

External Job
This type of job allows you to record the execution of a process run outside Jenkins, even on a remote machine. This is designed so that you can use Jenkins as a dashboard of your existing automation system. See [the documentation for more details](#).

Multi-configuration project
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

Setting up jenkins

- Add git repository to the project

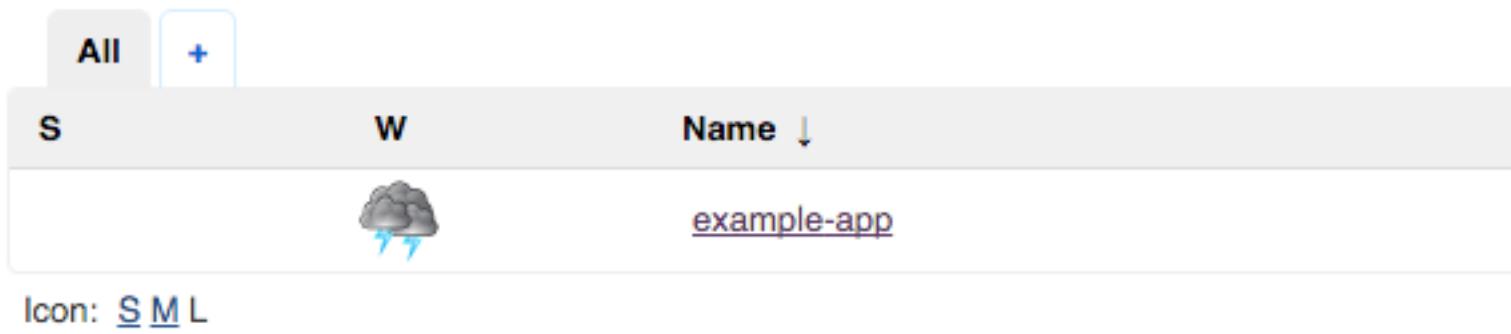
The screenshot shows the 'Source Code Management' section of a Jenkins job configuration. Under 'Repositories', 'Git' is selected. The 'Repository URL' field contains 'https://github.com/wardviaene/play-demo'. Below it, the 'Credentials' dropdown is set to '- none -' and there is an 'Add' button. On the right side, there are 'Advanced...', 'Add Repository', and 'Delete Repository' buttons.

- Add a build step

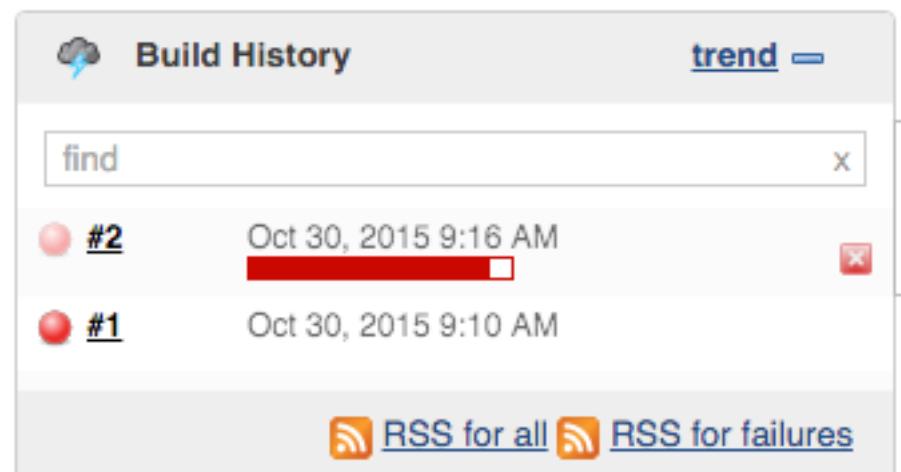
The screenshot shows the 'Build' section of a Jenkins job configuration. A dropdown menu on the left shows 'Build using sbt' selected. The main area displays the 'Build using sbt' configuration, including 'sbt launcher' set to 'sbt', 'JVM Flags' (empty), 'sbt Flags' containing '-Dsbt.log.noformat=true', and an 'Actions' section (empty). On the right, there are 'Advanced...', 'Delete' (in a red box), and other standard Jenkins buttons.

Build the project

- Click on the example-app link on the main page



- On the left we can start building the app by clicking “Build Now”:  Build Now
- To see the output, click on the blue or red bar in the Build History:

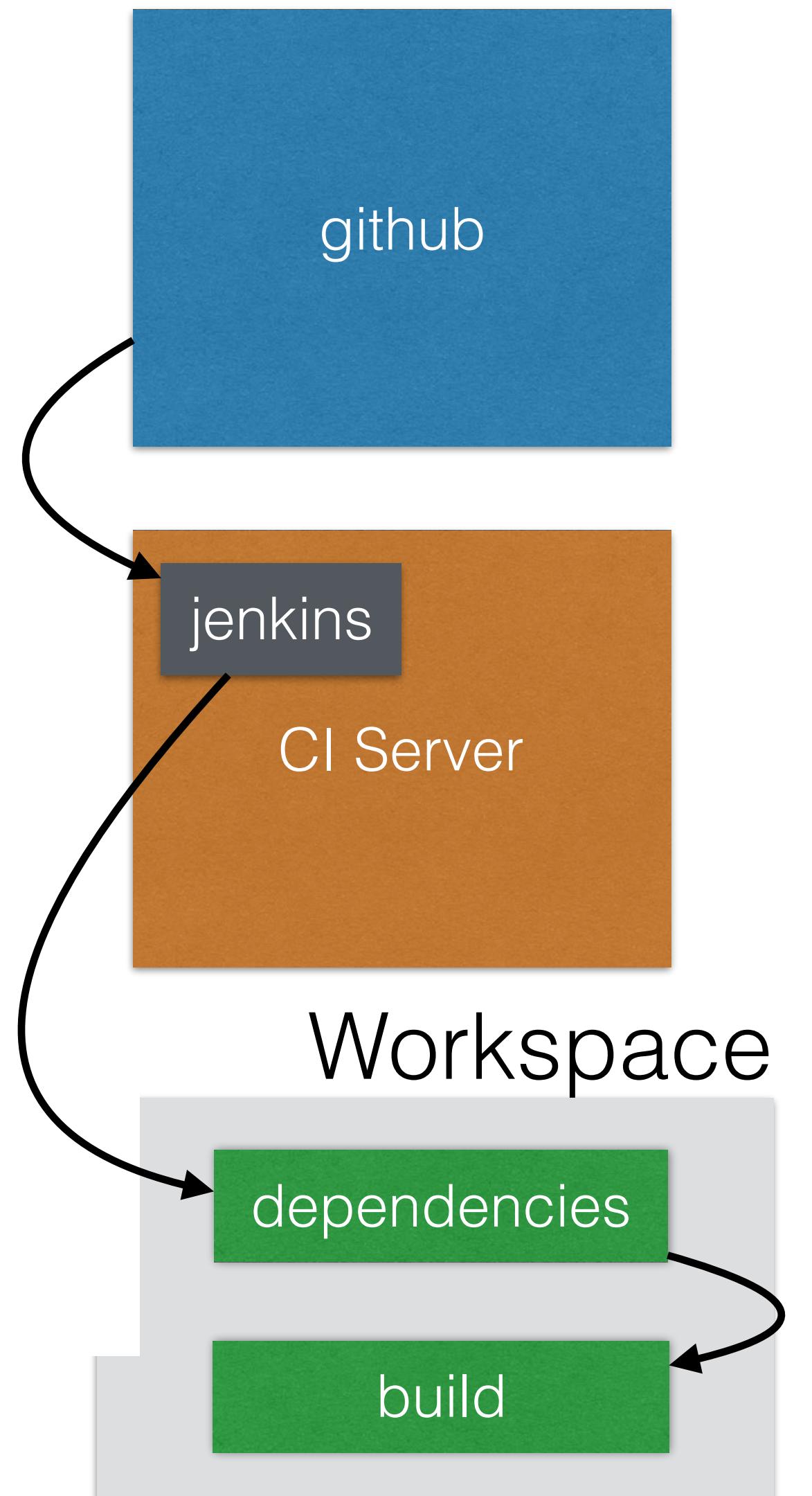


Build the project

- The project is now building
- Jenkins pulls the source from github and creates a new workspace
- sbt will get the necessary dependencies and build the project

```
Started by user anonymous
Building in workspace /var/lib/jenkins/jobs/example-app/workspace
> git rev-parse --is-inside-work-tree # timeout=10
Fetching changes from the remote Git repository
> git config remote.origin.url https://github.com/wardviaene/play-demo # timeout=10
Fetching upstream changes from https://github.com/wardviaene/play-demo
> git --version # timeout=10
> git -c core.askpass=true fetch --tags --progress https://github.com/wardviaene/play-demo +refs/heads/*:refs/remotes/origin/*
> git rev-parse refs/remotes/origin/master^{commit} # timeout=10
> git rev-parse refs/remotes/origin/origin/master^{commit} # timeout=10
Checking out Revision 354806c585c726974f51a4ec0c4e8b27b2921658 (refs/remotes/origin/master)
> git config core.sparsecheckout # timeout=10
> git checkout -f 354806c585c726974f51a4ec0c4e8b27b2921658
> git rev-list 354806c585c726974f51a4ec0c4e8b27b2921658 # timeout=10
[workspace] $ /usr/lib/jvm/java-8-oracle//bin/java -Xmx512M -Dsbt.log.noformat=true -jar /usr/share/sbt-launcher-packaging/bin/sbt-launch.jar
Getting org.scala-sbt sbt 0.13.8 ...
```

- The build can take a while



Build the project

- When the build is successfully completed, you should see a success message:

```
[info] Done updating.  
[info] Set current project to example-app (in build file:/var/lib/jenkins/jobs/example-app/workspace/)  
[example-app] $ Build step 'Build using sbt' changed build result to SUCCESS  
Finished: SUCCESS
```

- This means that the application was compiled and build successfully. Nothing broke our build.
 - It is possible that a developer might make some changes and breaks the build
 - It is good practice to have nightly build, or build when a developer pushes code

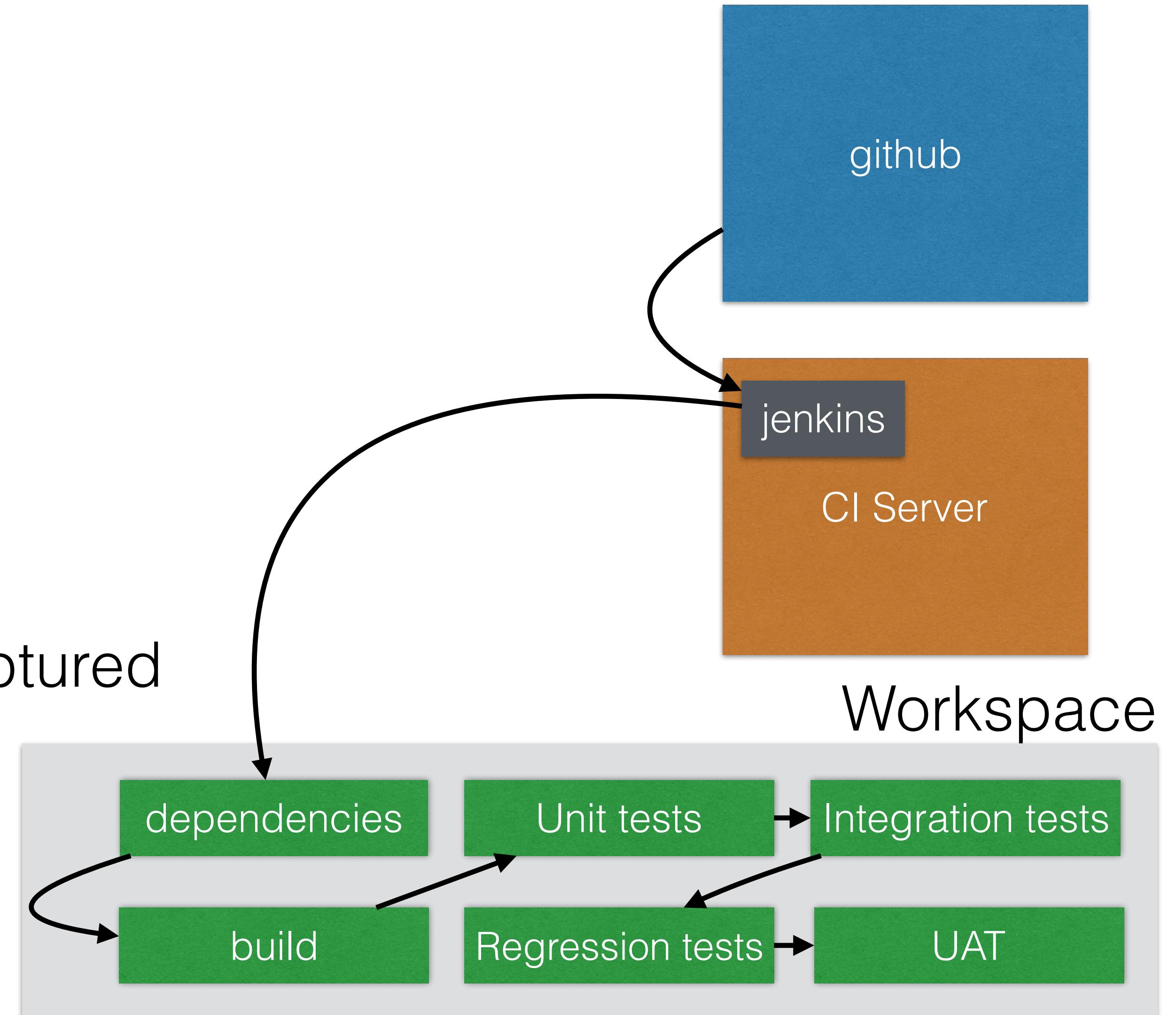
Jenkins demo

Automate testing

- The point of automating the build is to automate testing as well.
- Jenkins or any CI can run the tests for you and inform you of any failures
- It's still up to the developers to write good tests, if developers don't write tests, nothing will be tested

Common tests

- Unit tests
- Integration tests
- Regression tests
- User Acceptance Test (UAT)
- Manual, but approval can be captured



Implementing Unit tests

- Example test in java, stored in test folder:

```
public class ApplicationTest {  
  
    @Test  
    public void simpleCheck() {  
        int a = 1 + 1;  
        assertEquals(2, a);  
    }  
  
    @Test  
    public void renderTemplate() {  
        Content html = views.html.index.render("Your new application is ready.");  
        assertEquals("text/html", contentType(html));  
        assertTrue(contentAsString(html).contains("Your new application is ready."));  
    }  
}
```

Unit test

Implementing Integration tests

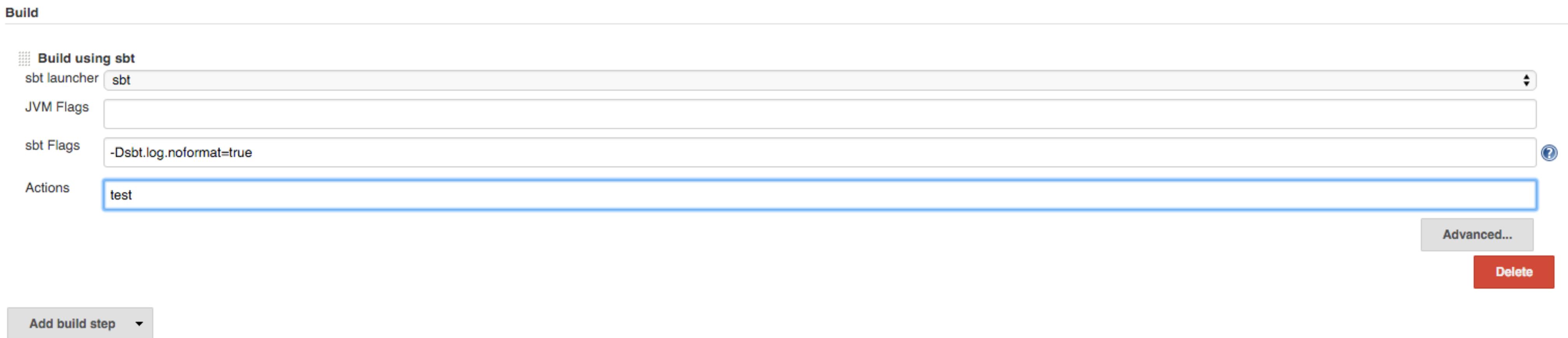
- Example test in java:

```
public class IntegrationTest {  
  
    /**  
     * add your integration test here  
     * in this example we just check if the welcome page is being shown  
     */  
    @Test  
    public void test() {  
        running(testServer(3333, fakeApplication(inMemoryDatabase())), HTMLUNIT, new Callback<TestBrowser>() {  
            public void invoke(TestBrowser browser) {  
                browser.goTo("http://localhost:3333");  
                assertTrue(browser.pageSource().contains("Your new application is ready."));  
            }  
        });  
    }  
}
```

Integration test

Adding unit tests to jenkins

- To enable tests in jenkins, you'll need to change the configuration of the project
 - Click Configure in the example-app:  [Configure](#)
 - Alternatively go to <http://192.168.0.252:8080/job/example-app/configure>
- Scroll down to build and add “test” to Actions:

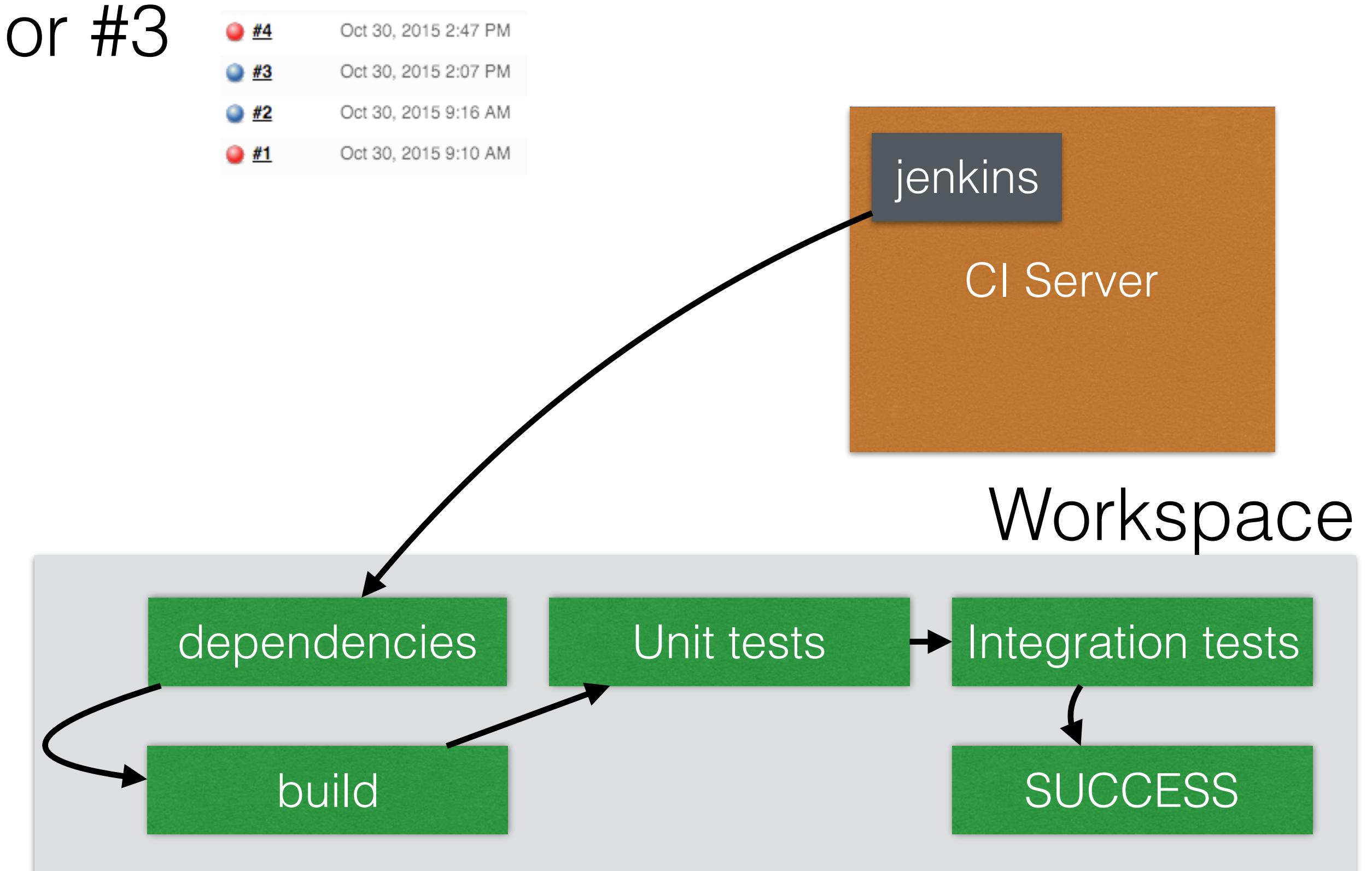


The screenshot shows the Jenkins configuration interface for a job named "example-app". The "Build" section is visible, containing fields for "Build using sbt" (with "sbt launcher" set to "sbt"), "JVM Flags" (empty), "sbt Flags" (-Dsbt.log.noformat=true), and an "Actions" section. In the "Actions" section, the word "test" is entered into the text input field, which is highlighted with a blue border. Below the input field are buttons for "Advanced..." and "Delete". At the bottom left, there is a "Add build step" button.

Adding unit tests to jenkins

- Rebuild the application by clicking “rebuild now”
- Click on the build number (#2 or #3) and click on the “console output link” or click on the blue bar next to #2 or #3
- The output should now look like:

```
[info] Passed: Total 3, Failed 0, Errors 0, Passed 3
[success] Total time: 307 s, completed Oct 30, 2015 2:13:12 PM
Build step 'Build using sbt' changed build result to SUCCESS
Finished: SUCCESS
```

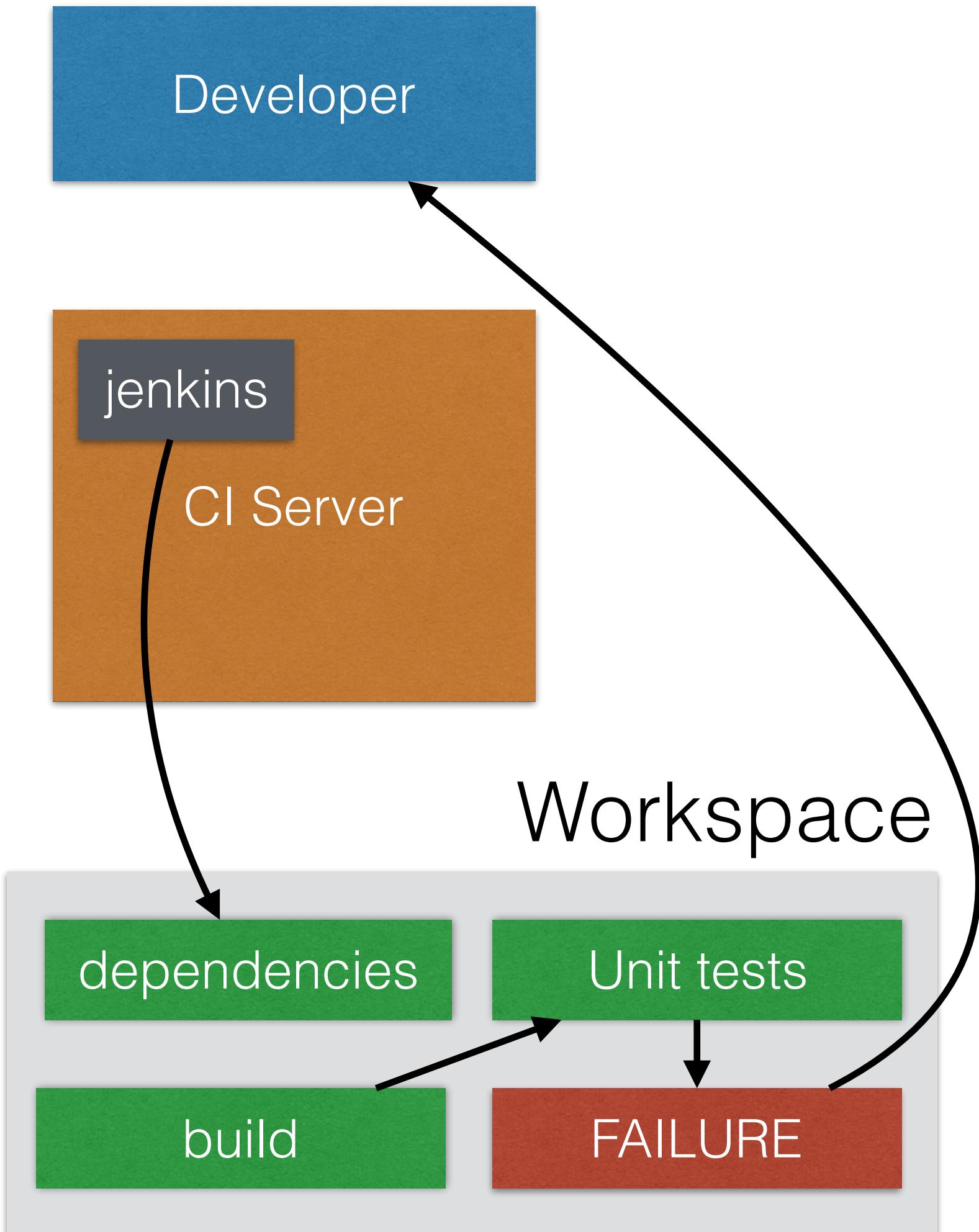


Implementing Unit tests

- When would a test fail?
- Imagine a test like:

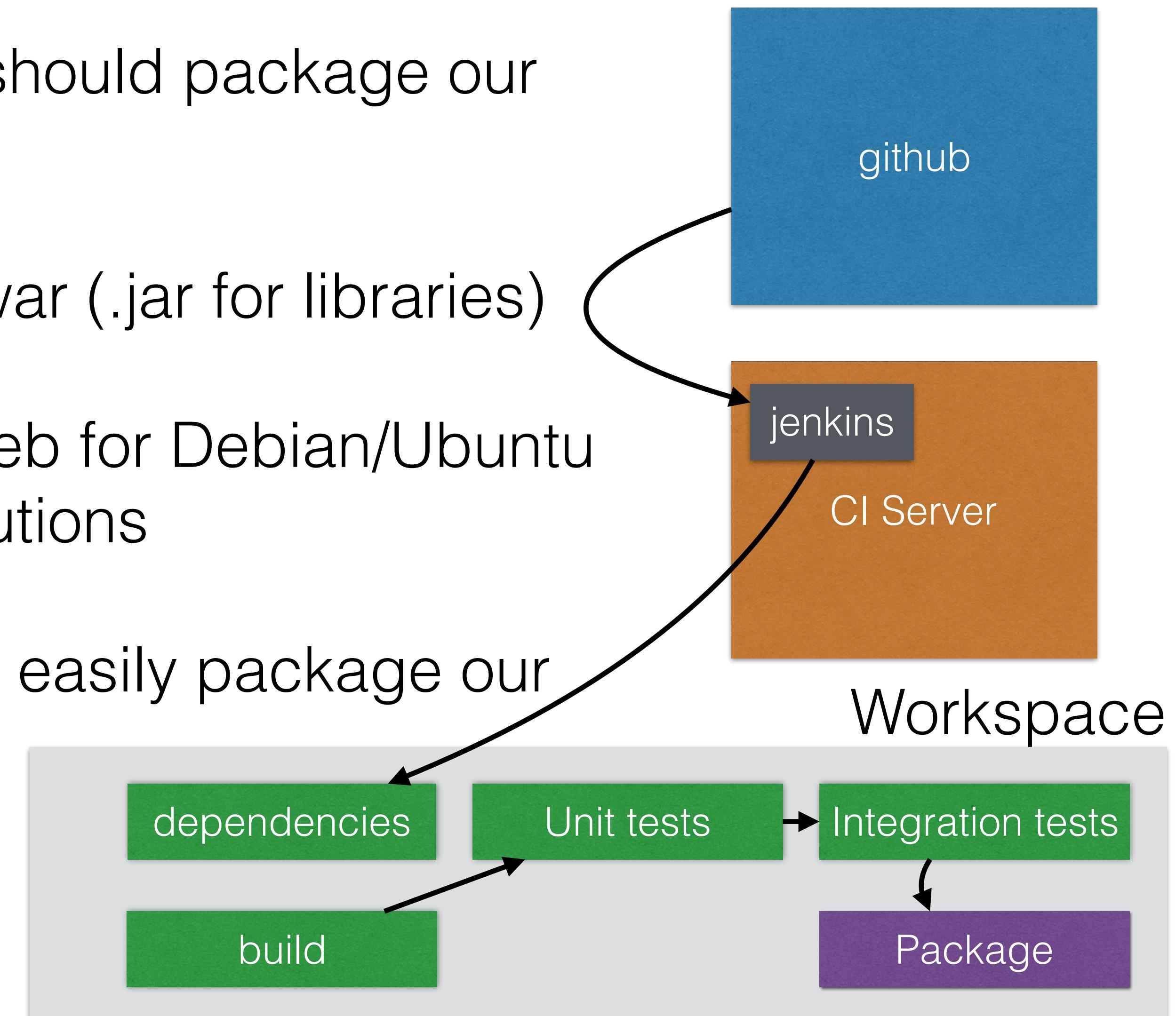
```
public class ApplicationTest {  
    @Test  
    public void simpleCheck() {  
        int a = Sum(1, 2);  
        assertEquals(3, a);  
    }  
}
```

- If our method `Sum(1,2)` would multiply numbers instead of adding together, `a` would be equal to 2 instead of 3
- `assertEquals(3, a)` would then fail, so would our sbt test in jenkins



Packaging the project

- To make deployments easier, we should package our newly created website
- Java packages are often .zip or .war (.jar for libraries)
- Linux installation packages are .deb for Debian/Ubuntu and .rpm for redhat-based distributions
- Using the play framework, we can easily package our website as .deb



Packaging the project

- Extra code is necessary to load the debian plugin to build the .deb and minimal information that the packager needs
- Packaging code in build.sbt

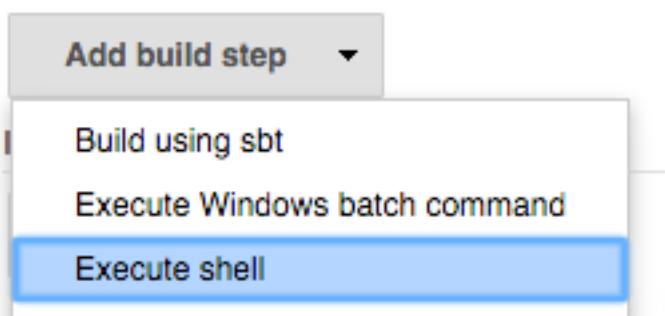
```
enablePlugins(DebianPlugin)
maintainer := "Your Name <email@address>"
packageSummary := "My custom package"
packageDescription := "Package"
```

- git commit & push

```
vagrant@jenkins:~/example-app$ git commit -am "added debian package info"
vagrant@jenkins:~/example-app$ git push
vagrant@jenkins:~/example-app$
```

Packaging the project

- Let's change the configuration again. This time, we are going to add a shell script to the build:



- The following command executes the activator in the project directory and builds a debian package
- jenkins calls it the artifact:

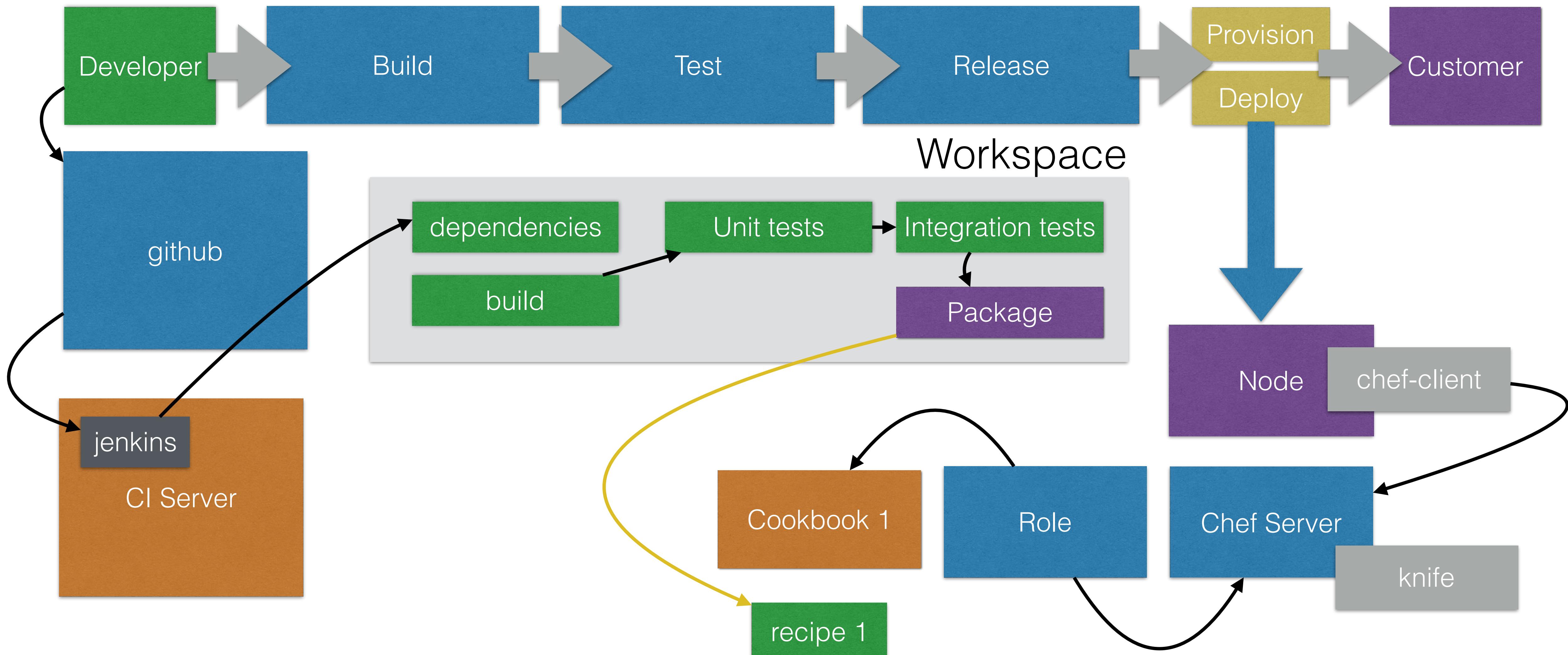
The screenshot shows the Jenkins configuration for an 'Execute shell' build step. The 'Command' field contains the command: `./activator debian:packageBin`. Below the command field, there is a link to 'See the list of available environment variables' and a red 'Delete' button.

Jenkins demo

automate testing

Deployment

SDLC: Deployment



Artifact storage

- We want to store the artifacts generated by our build process
 - jar (java archive) to use as libraries
 - deb/rpm files to install on our servers
 - generic zip/tar files
- There is software available, but most of them are commercial:
 - Artifactory
 - Nexus
 - Archiva

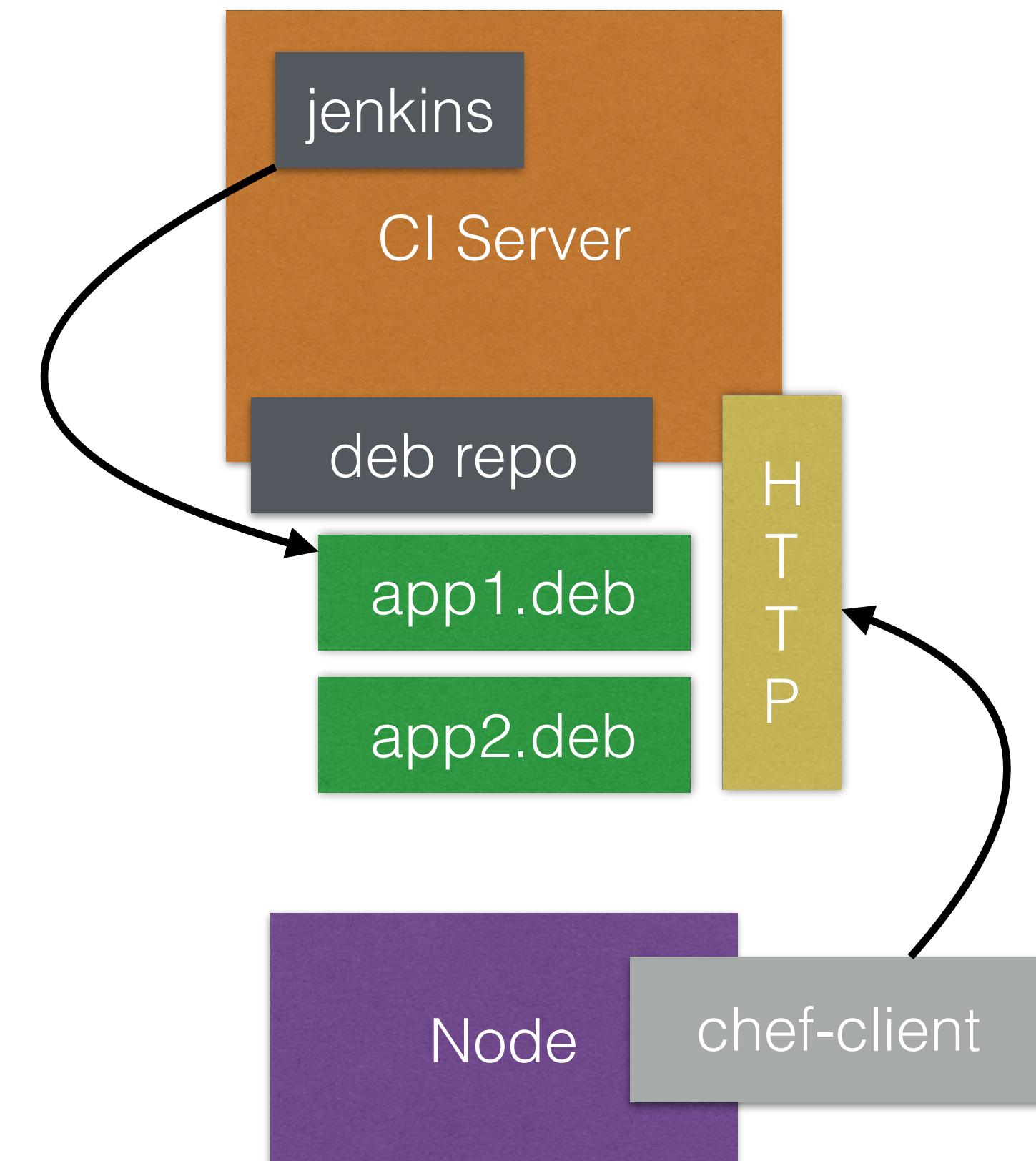
Artifact storage

- An debian based repository can be created yourself (with some work)
- First, create a new gpg key:

```
vagrant@jenkins:~$ sudo su - jenkins  
jenkins@jenkins:~$ gpg —gen-key  
jenkins@jenkins:~$
```

- if gpg has trouble finding random entropy, execute the following commands to generate random data

```
vagrant@jenkins:~$ sudo apt-get install rng-tools  
vagrant@jenkins:~$ sudo rngd -r /dev/urandom  
vagrant@jenkins:~$
```



Artifact storage

- Get the signing key ID and public key id (you will need it later)
 - For repository configuration

```
vagrant@jenkins:~$ sudo su - jenkins
jenkins@jenkins:~$ gpg --list-keys
/var/lib/jenkins/.gnupg/pubring.gpg
-----
pub 2048R/8D53BAA5 2015-10-30
uid          gpg key for deb (gpg key for deb) <ward@in4it.io>
sub 2048R/06EFF9F7 2015-10-30
```

- Send public key to ubuntu key server, to easily retrieve it later

```
jenkins@jenkins:~$ gpg --keyserver keyserver.ubuntu.com --send-key 06EFF9F7
gpg: sending key 8D53BAA5 to hkp server keyserver.ubuntu.com
jenkins@jenkins:~$
```

Artifact storage

- Install nginx and reprepro, then create a repository (replace the key in blue)

```
vagrant@jenkins:~$ sudo apt-get install nginx reprepro
vagrant@jenkins:~$ sudo -s
root@jenkins:~# mkdir -p /var/repositories/
root@jenkins:~# cd /var/repositories/
root@jenkins:/var/repositories# mkdir conf
root@jenkins:/var/repositories# cd conf/
root@jenkins:/var/repositories/conf# touch options distributions
root@jenkins:/var/repositories/conf# cat options
root@jenkins:/var/repositories/conf# cat << EOF > distributions
> Codename: trusty
> Components: main
> Architectures: i386 amd64
> SignWith: 06EFF9F7
> EOF
root@jenkins:/var/repositories/conf# chown -R jenkins:jenkins /var/repositories/
root@jenkins:/var/repositories/conf#
```

Artifact storage

- Next, configure nginx:

```
vagrant@jenkins:~$ sudo -s
root@jenkins:~# cat << EOF > /etc/nginx/sites-available/default
> server {
>   ## Let your repository be the root directory
>   root      /var/repositories;
>   access_log /var/log/nginx/repo.access.log;
>   error_log  /var/log/nginx/repo.error.log;
>
>   ## Prevent access to Repro's files
>   location ~ /(dblconf) {
>     deny    all;
>     return  404;
>   }
> }
> EOF
root@jenkins:~# service nginx restart
root@jenkins:~#
```

Artifact storage

- Let's add a post build action to jenkins
- Edit your project configuration at <http://192.168.0.252:8080/job/example-app/configure>

The screenshot shows the Jenkins build step configuration interface. A dropdown menu is open under 'Add build step' with the following options: 'Build using sbt', 'Execute Windows batch command', 'Execute shell' (which is selected and highlighted in blue), 'Invoke Ant', 'Invoke top-level Maven targets', and 'Play!'. Below this, a specific 'Execute shell' step is configured with the command: 'reprepro -b /var/repositories remove trusty example-app; reprepro -b /var/repositories includedeb trusty ./target/example-app_*.deb'. There are links for 'See the list of available environment variables' and a 'Delete' button.

- And run the build again

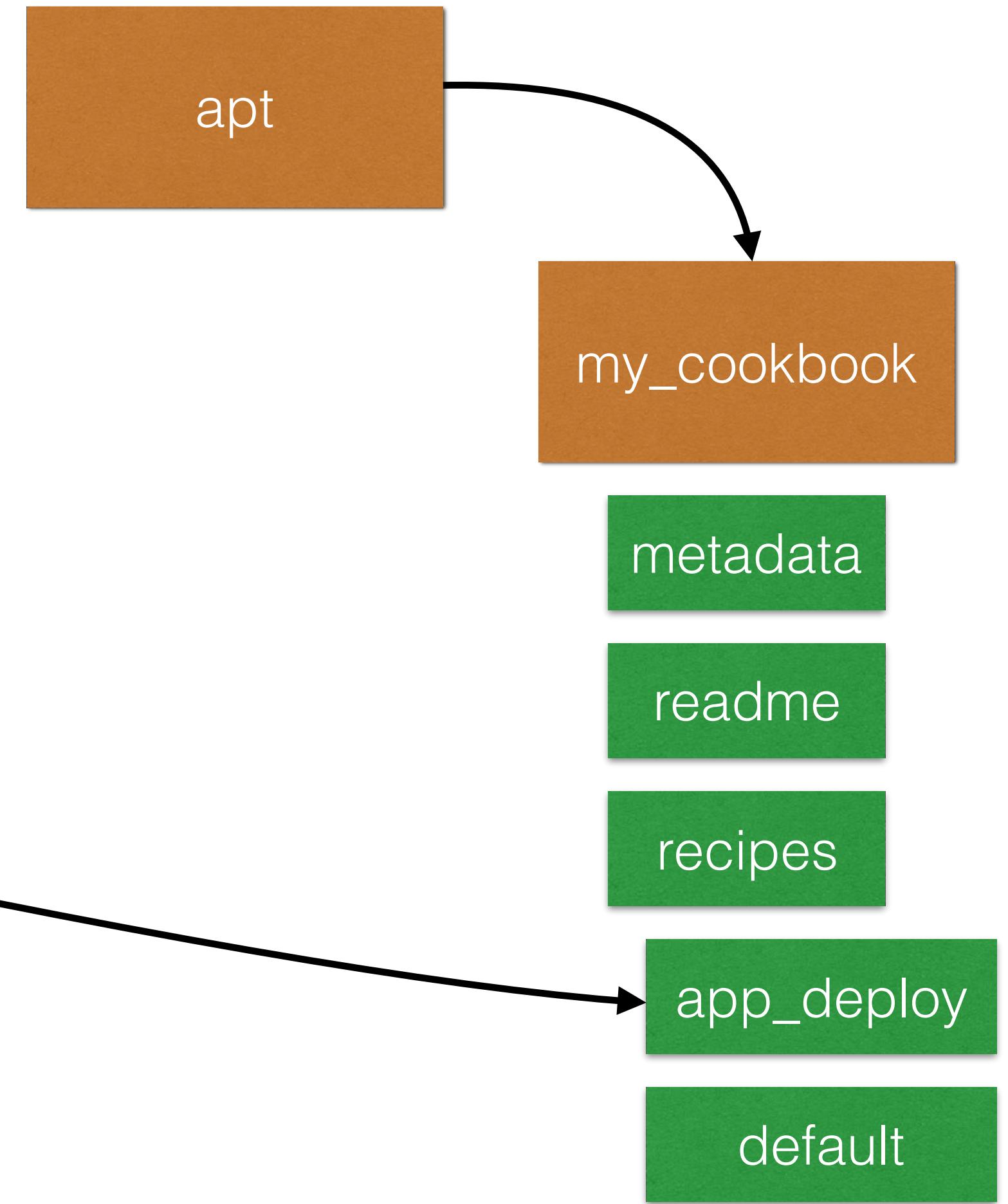
Artifact storage

- The .deb file is added to the apt repository we created
 - We could run apt-get update && apt-get install example-app
 - A better way is to automate the deployment using chef / ansible

Deploy app

- New recipe in chef my_cookbook/recipes/app_deploy.rb :

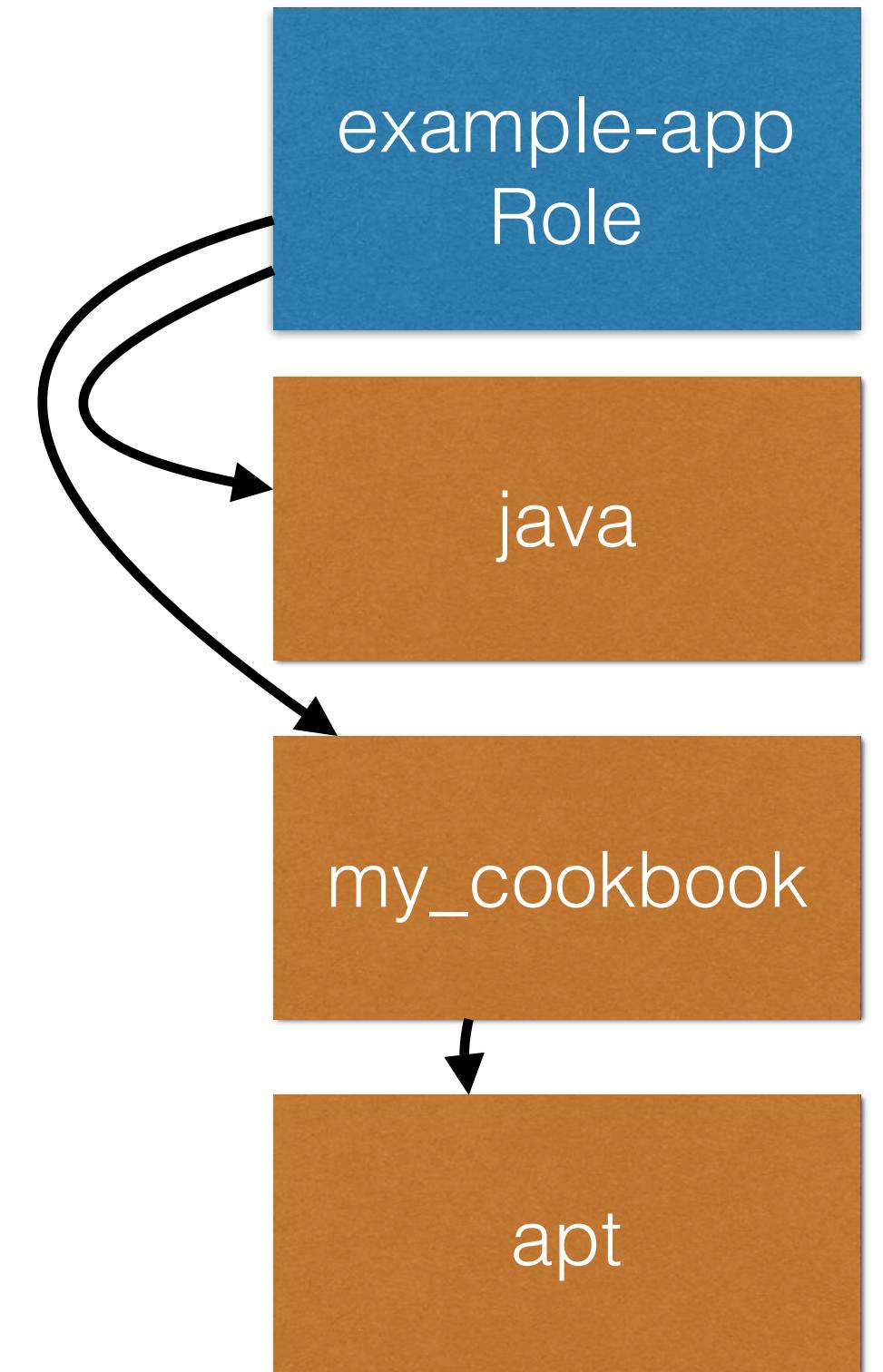
```
include_recipe "apt"  
apt_repository "example-app" do  
  uri "http://192.168.0.252"  
  distribution "trusty"  
  components ["main"]  
  keyserver "keyserver.ubuntu.com"  
  key "8D53BAA5"  
end  
package 'example-app' do  
  action :install  
end  
directory '/usr/share/example-app/' do  
  owner 'example-app'  
  group 'example-app'  
  recursive true  
end  
service 'example-app' do  
  provider Chef::Provider::Service::Upstart  
  action [:enable, :start]  
end
```



Deploy app

- new example-app.json role

```
{  
  "name": "example-app",  
  "default_attributes": {  
    "java": {  
      "install_flavor": "oracle",  
      "jdk_version      "oracle": {  
        "accept_oracle_download_terms": true  
      }  
    }  
  },  
  "json_class": "Chef::Role",  
  "run_list": ["recipe[java]", "recipe[my_cookbook::app_deploy]"],  
  "description": "example app role that installs my_cookbook",  
  "chef_type": "role",  
  "override_attributes": {}  
}
```



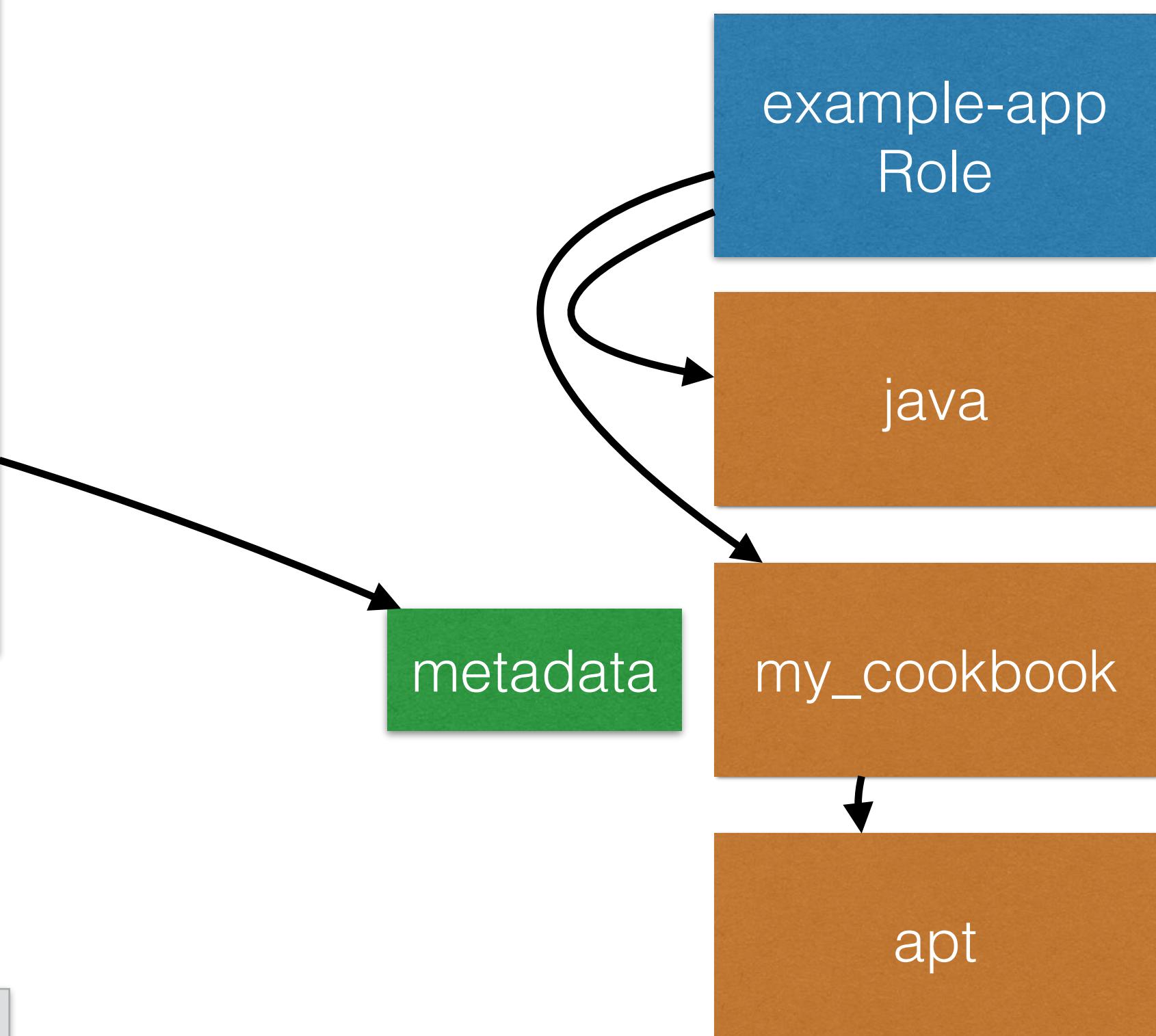
Deploy app

- metadata file: add java, increment version number

```
name      'my_cookbook'  
maintainer      'YOUR_COMPANY_NAME'  
maintainer_email 'YOUR_EMAIL'  
license      'All rights reserved'  
description    'Installs/Configures test'  
long_description IO.read(File.dirname(__FILE__), 'README.md'))  
version      '0.2.9'  
depends       'apt'  
depends      'java'
```

- run berks install again, to download the apt and java cookbook

```
vagrant@chef:~/cookbooks$ BERKSHELF_PATH=~/ berks install
```



Deploy app

- Upload cookbooks

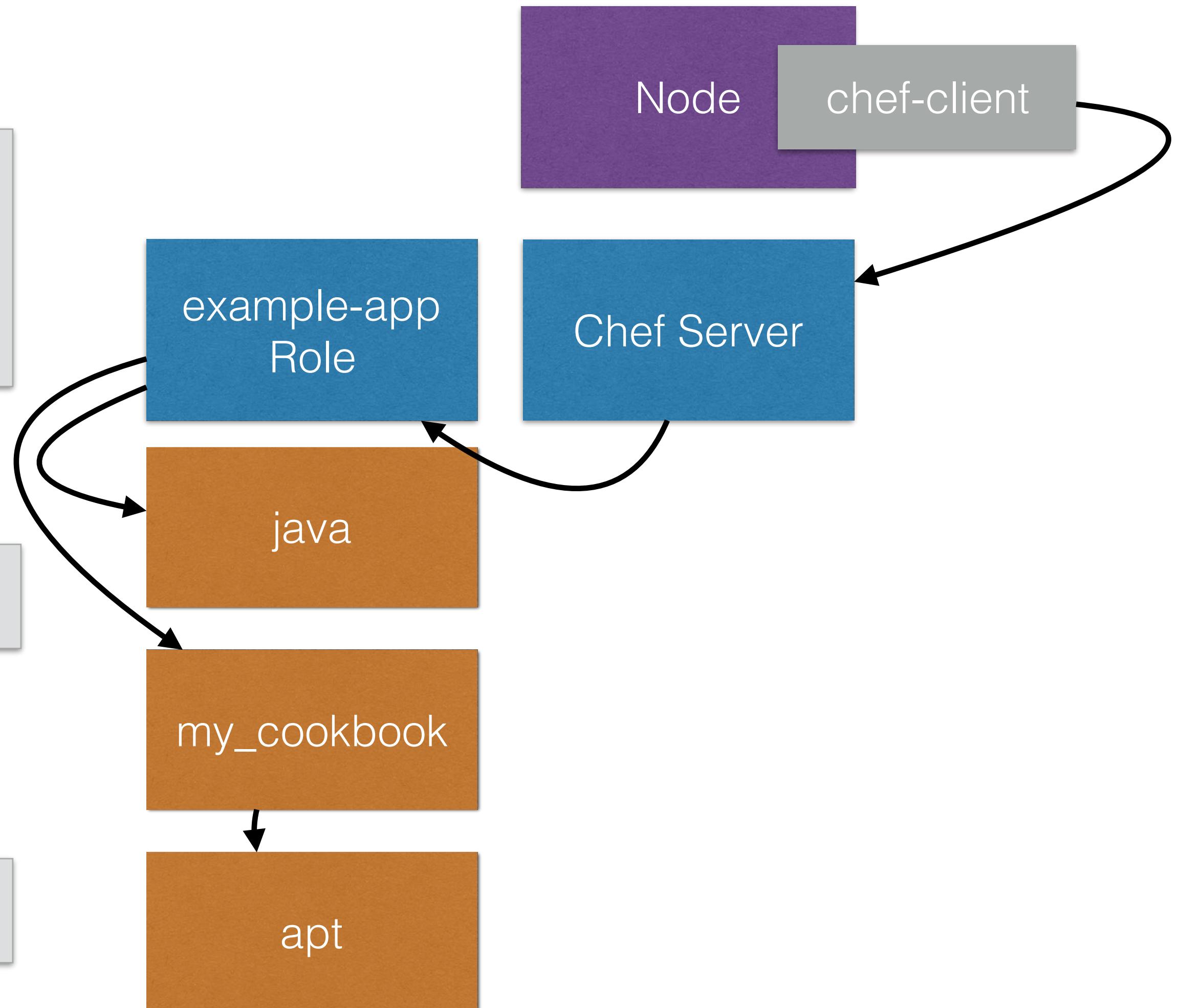
```
vagrant@chef:~/cookbooks$ knife cookbook upload apt  
vagrant@chef:~/cookbooks$ knife cookbook upload java  
vagrant@chef:~/cookbooks$ knife cookbook upload my_cookbook  
vagrant@chef:~/cookbooks$ knife role from file example-app.json
```

- change role in node's run_list:

```
vagrant@chef:~/cookbooks$ knife node run_list set node 'role[example-app]'
```

- run chef-client

```
vagrant@chef:~/cookbooks$ ssh node 'sudo chef-client'
```



App deployed



Your new application is ready.

Deployment using Ansible

- The same can be achieved using ansible:
 - using apt_repository module

```
apt_repository: repo='deb http://192.168.0.252 trusty main' state=present update_cache=yes
```

- Adding the app using apt:

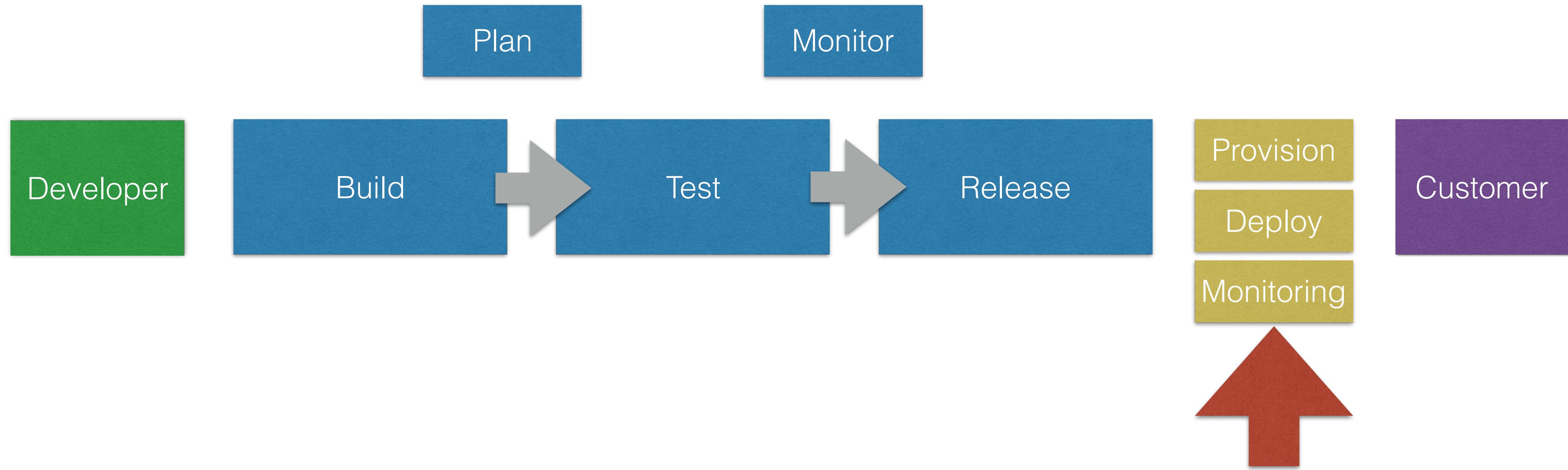
```
- apt: name=example-app
```

Artifact storage and Deployment

demo

Continuous Monitoring

SDLC



- When provisioning and deploying, infrastructure should be monitored as well.
- Automation of monitoring: Continuous Monitoring

Nagios

- Nagios is often used for monitoring
- Initial release 1999: old, but very mature
- A lot of plugins available
- Agents available that can be installed on clients
- Alternatives: sensu, zabbix, AWS CloudWatch

Application Monitoring

- Application Performance Monitoring (APM)
- First line of monitoring, can give you instant alerts when an application “slows down”
- New Relic is the most popular one
- Application specific

Log File Aggregation

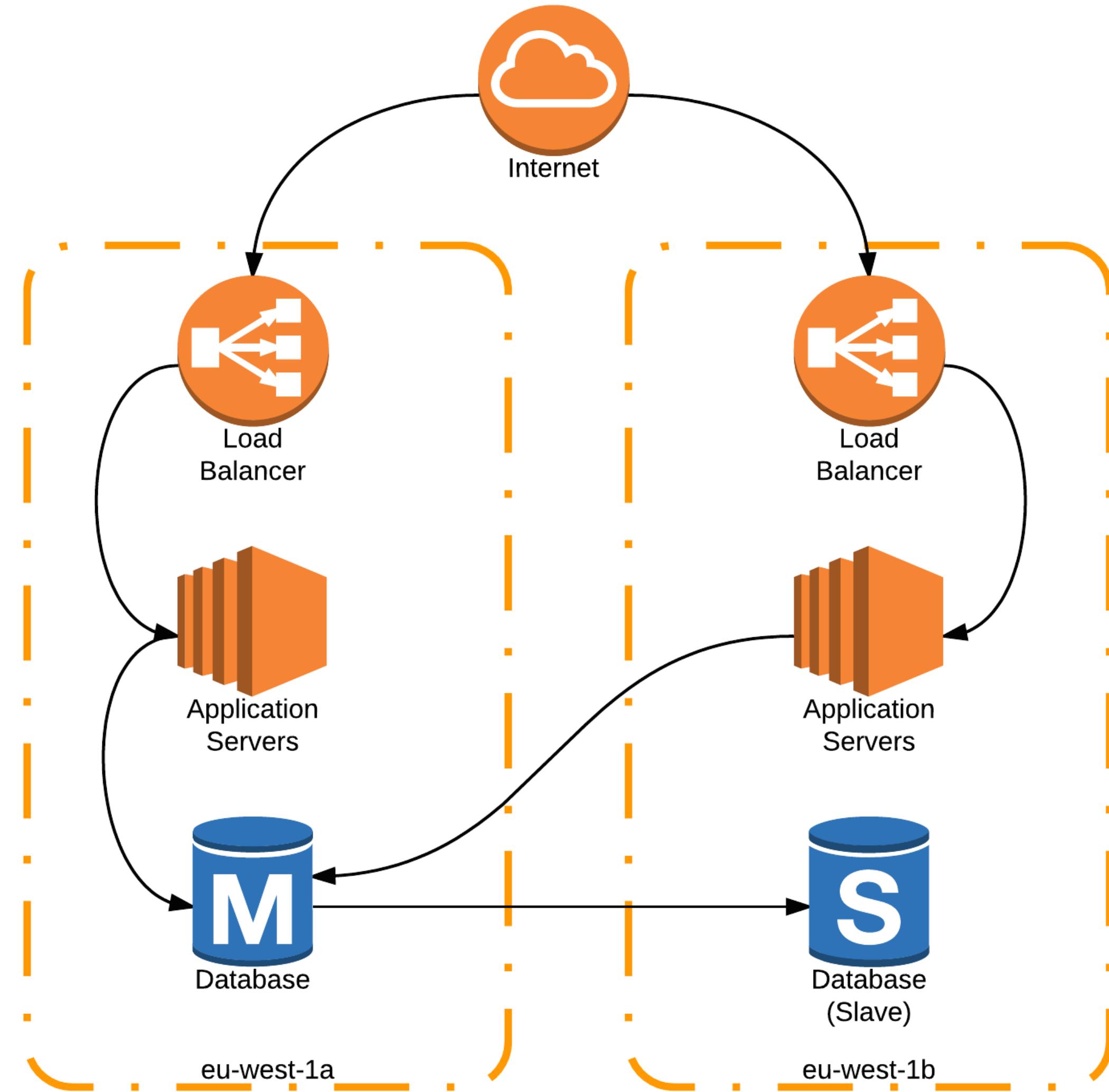
- Log files can give a good insight of what is happening on servers
- ELK stack (Elasticsearch Logstash Kibana) can scale over thousands of servers
- It's more work to setup
- Complex algorithms can be ran over log output
- Visualization can give you good insights

Twelve Factor App

The Twelve Factor App

- Nowadays, software is commonly delivered as a service
- 12-Factor is a methodology for building software as a service apps that:
 - Have a declarative format for setup automation
 - Have Maximum portability
 - Are suitable for modern Cloud Platforms
 - Minimize divergence between dev and prod
 - Can scale

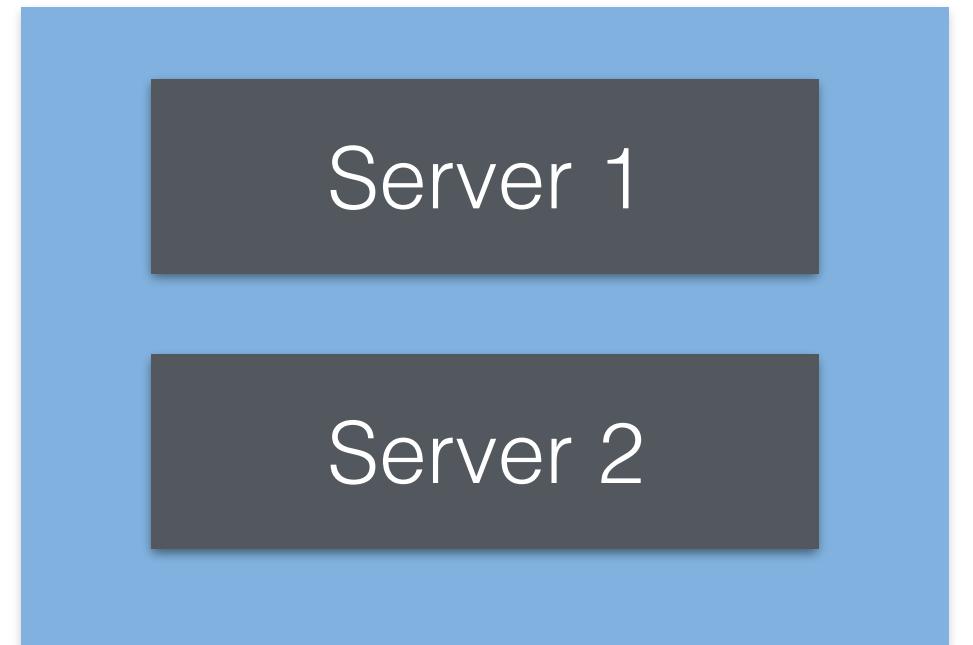
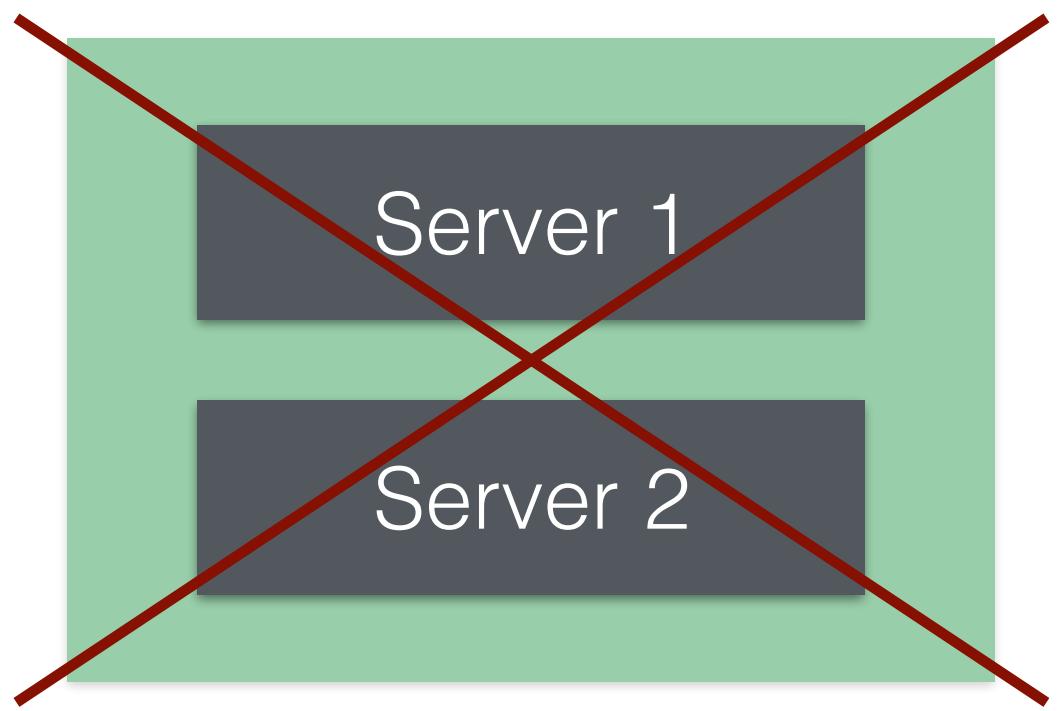
Example architecture



- Amazon AWS Deployment
- 2 zones for HA
- DNS is used to point to the 2 Load Balancers
- Load Balancers use health checks to determine health of application servers
- 1 DBaaS, which is replicated over 2 zones

Advantages

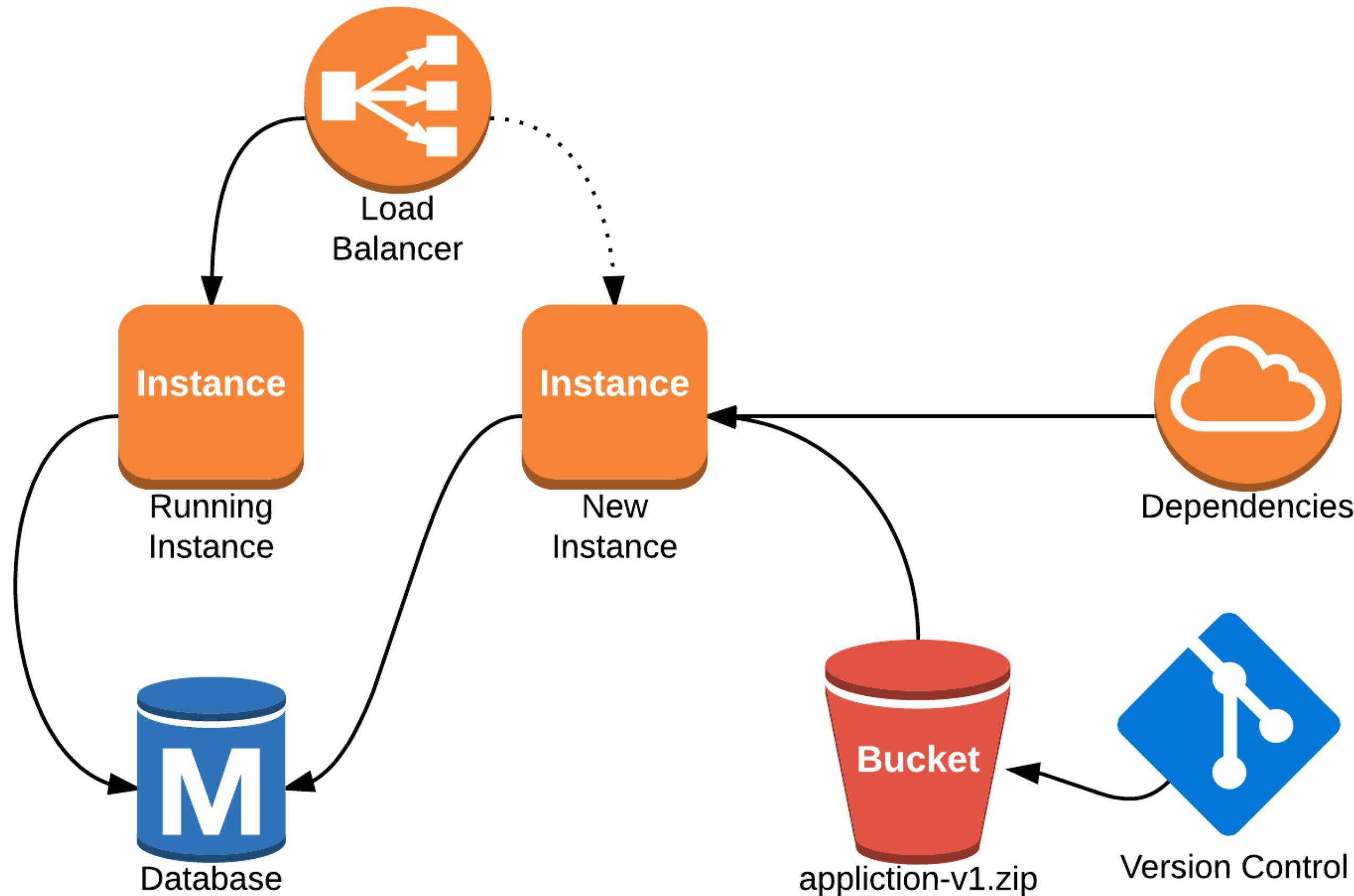
- Decouple application from infrastructure / OS
- Enables Green/Blue deployments
- Zero downtime deployments and upgrades
- Works well with Continuous Delivery & Continuous Deployments
- Best Practice



Advantages (cont.)

- Avoid Software Erosion
 - Slow deterioration of software over time that will eventually lead to the software becoming faulty or unusable
 - The software does not actually decay, but rather suffers from a lack of being updated with respect to the changing environment in which it resides

Deployment



- New release gets archived from version control or gets build from CI
- New instance starts
- New app version gets deployed on a new instance
- Application Dependencies are retrieved
- Application is live, instance is now healthy
- Load balancer will add new instance and will send traffic to the newly deployed application
- Older instance can be upgraded or shut down

The 12 Factors

I. Codebase

- One codebase tracked in revision Control, many deploys

II. Dependencies

- Explicitly declare and isolate Dependencies

III. Config

- Store config in environment

IV. Backing Services

- Treat backing services as attached resources

The 12 Factors

V. Build, Release, Run

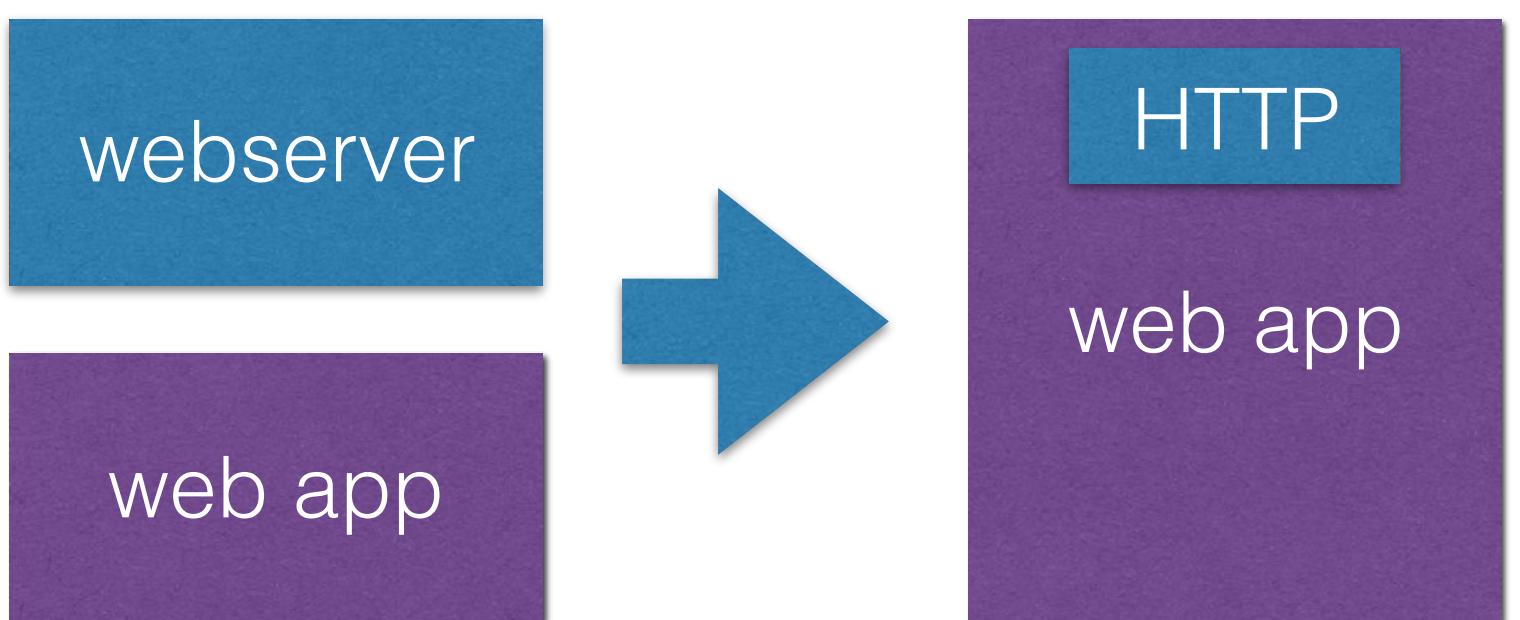
- Strictly separate build and run stages

VI. Processes

- Execute the app as one or more stateless processes

VII. Port binding

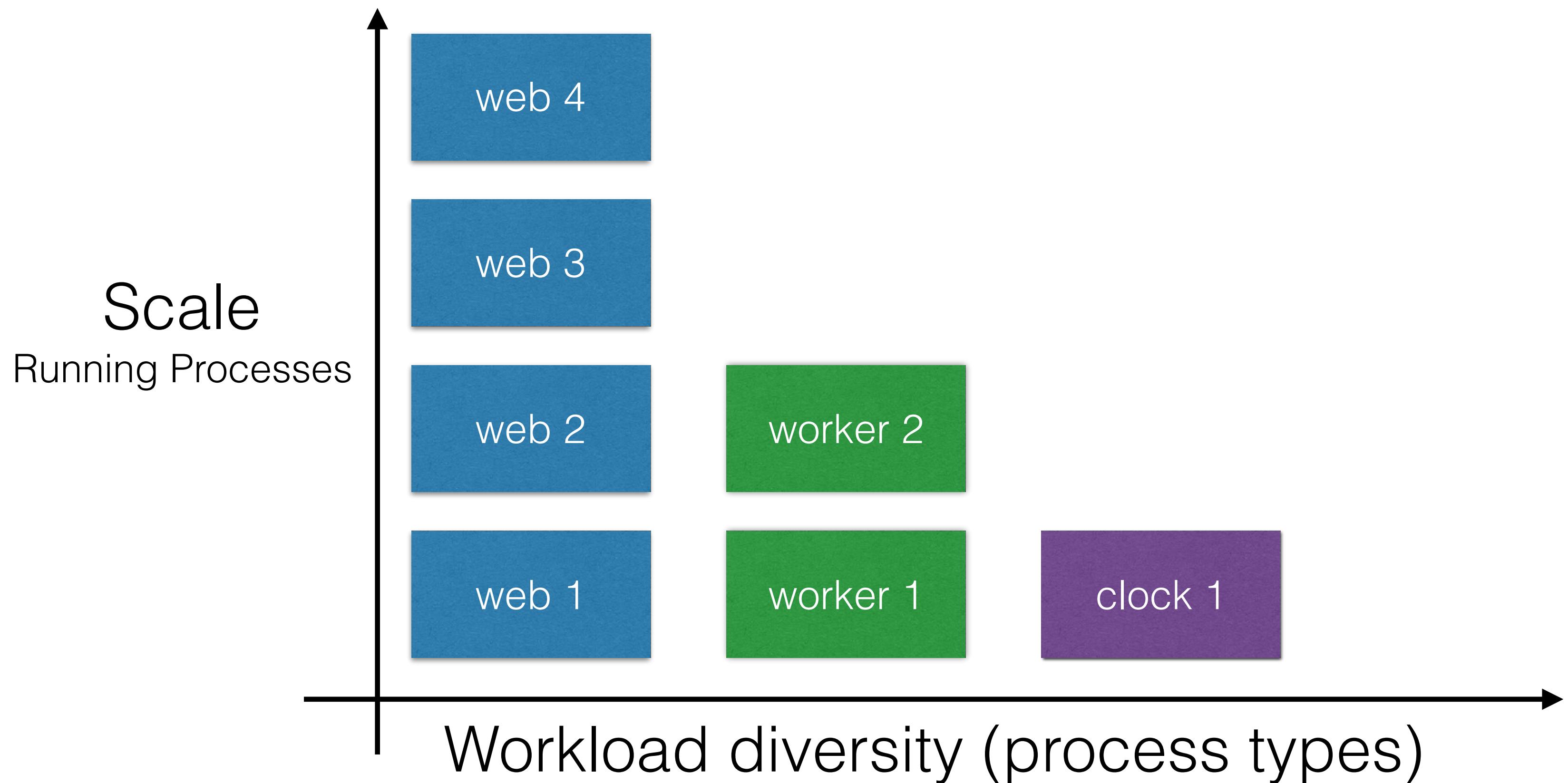
- Export services via port binding



The 12 Factors

VIII. Concurrency

- Scale out via the process model



The 12 Factors

IX. Disposability

- Maximize robustness with fast startup and graceful shutdown

X. Dev / Prod parity

- Keep development, staging, and production as similar as possible

XI. Logs

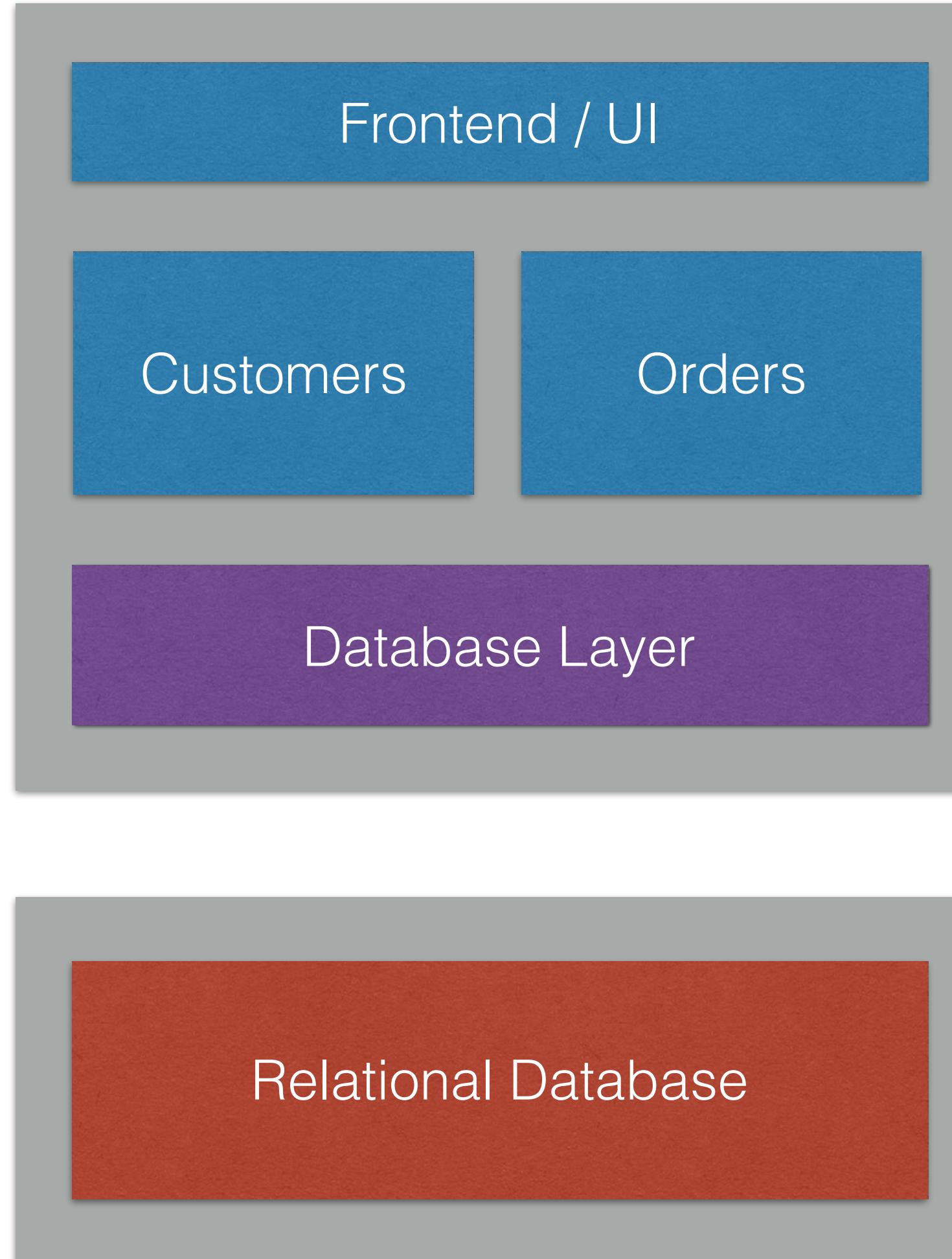
- Treat logs as event streams, apps should output logs to stdout

XII. Admin Processes

- Run admin/management tasks as one-off processes

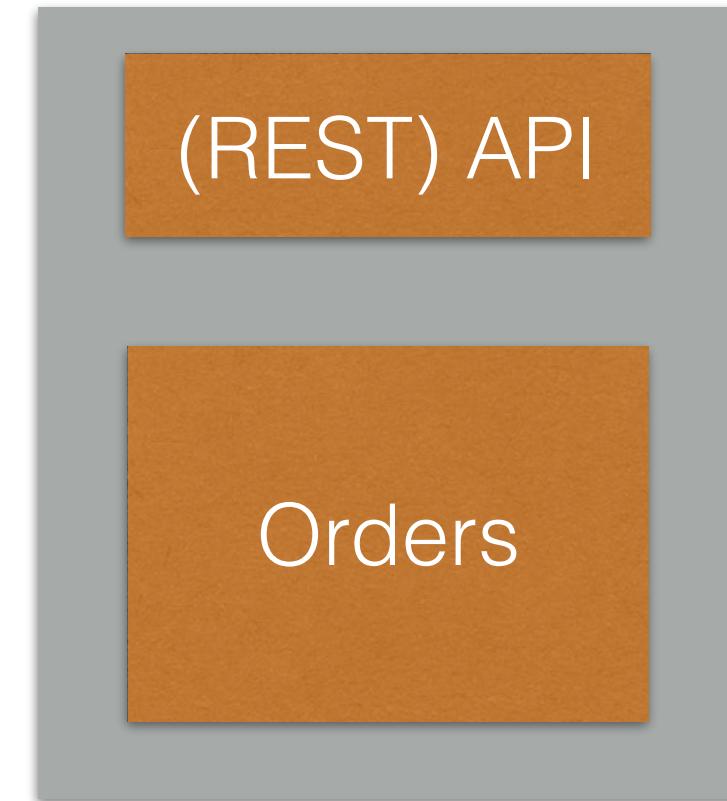
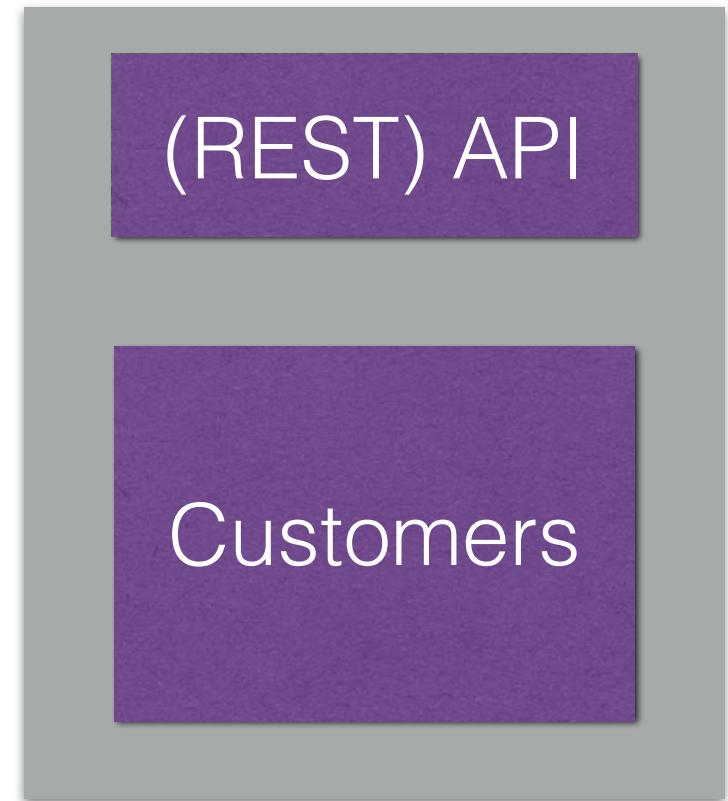
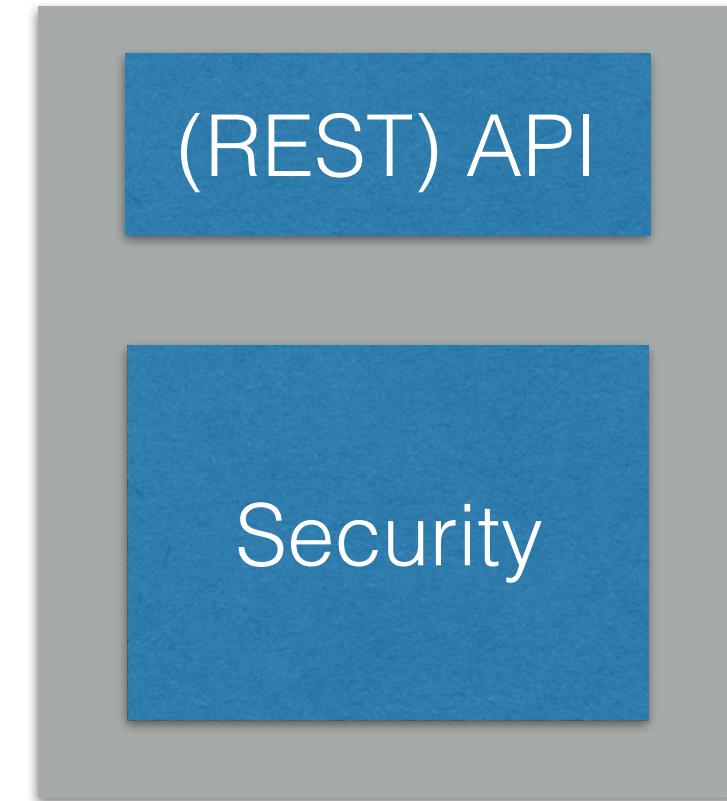
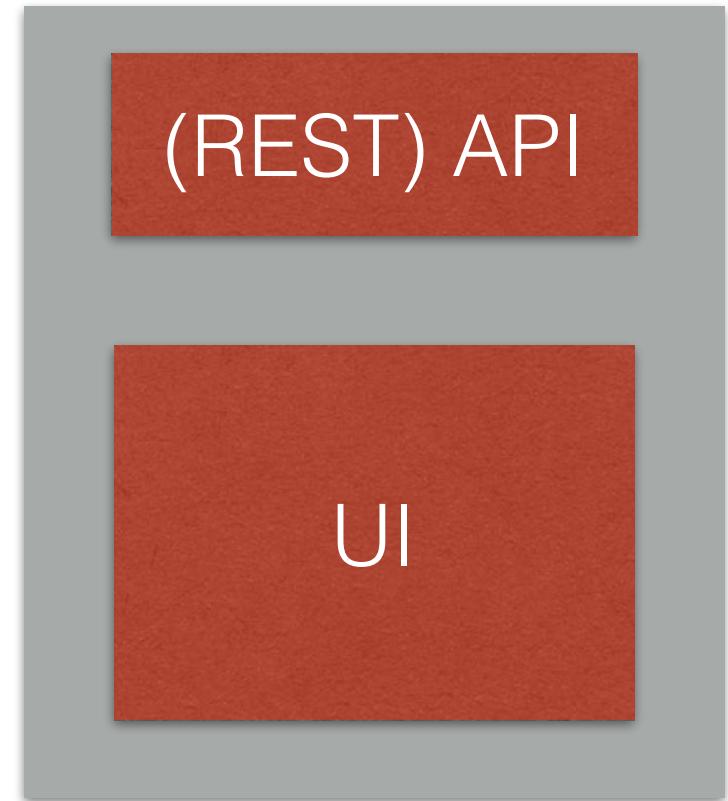
Microservices

The monolithic application



- Monolithic application
- Easy to develop, test and deploy
- Tends to become large and complex
- Difficult to work on as a team
- Higher risk of failure when deploying

Microservices



- Monolithic application divided into smaller “microservices”
- Smaller service can use its own technology stack
- Easier for developers to understand a single service
- Quicker to build and faster to deploy

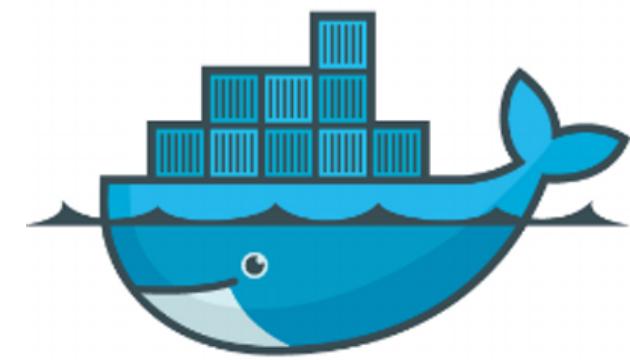
DevOps of Microservices

- DevOps of microservices needs a different approach
- The application becomes distributed
- Microservices can scale horizontal
 - More fault tolerant, scales quicker than vertical
- Virtual Machines (VMs) are often too slow, inefficient and too big to transfer
- ➔ Containerization is a better choice when adopting the microservice architecture

Containerization

Docker

- Open platform for developers and sysadmins to build, ship, and run distributed applications
 - Docker Engine: a portable, lightweight runtime and packaging tool
 - Docker Hub: a cloud service for sharing applications and automating workflows
- Enables apps to be quickly assembled from components
- Eliminates the friction between Dev, QA, and production environments
- It should be able to ship faster
- Should be able to run the same app, unchanged, on laptops, data center VMs, and cloud.
- Docker uses LXC (Linux Containers) for operating system-level virtualization



How to use Docker

- Using boot2docker
- Using Vagrant with Linux
- Using Cloud
 - Amazon AWS ECS
 - Google Container Engine

Boot docker using vagrant

- Spin up a new vagrant box:

```
$ vagrant init ubuntu/trusty64  
$ vagrant up  
$ vagrant ssh
```

- Install Docker

```
vagrant@docker:~$ sudo apt-get install docker.io
```

- After installing docker, it is immediately ready for use

Docker

- Use docker run to start a container and execute a command

```
vagrant@docker:~$ sudo docker run centos:7 /bin/echo 'Hello world'
```

- This command runs a container using CentOS 7
- If the image is not available, it will be downloaded automatically
- Once the container is started, the command /bin/echo 'Hello world' gets executed and the container stops

Docker

- To launch an interactive container, the following command can be executed:

```
vagrant@docker:~$ sudo docker run -t -i ubuntu:14.04 /bin/bash
root@af8bae53bdd3:/#
```

- -i runs the container in interactive mode
- /bin/bash starts the bash shell
- The container will run until we exit bash

Docker

- We can serve a web page from inside a container

```
vagrant@docker:~$ sudo docker run -p 127.0.0.1:8080:80 -i ubuntu:14.04 nc -kl 80
```

- -p maps the port from the container at port 80 to 8080 on the host machine

Dockerfile

- Docker can build images automatically by reading the instructions from a Dockerfile
- A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image
- Using docker build users can create an automated build that executes several command-line instructions in succession.

Dockerfile example

- Let's run a simple nodeJS project in docker
- Using the express framework, these lines in node show a page with "hello world"

```
var express = require('express');
var app = express();

app.get('/', function (req, res) {
  res.send('Hello World!');
});

var server = app.listen(3000, function () {
  var host = server.address().address;
  var port = server.address().port;

  console.log('Example app listening at http://%s:%s', host, port);
});
```

create a new directory called docker-demo and save the file as index.js

Dockerfile example

- packages.json determines the dependencies of our project
- It's part of node, and in our Dockerfile we'll use this to retrieve dependencies

```
{  
  "name": "myapp",  
  "version": "0.0.1",  
  "private": true,  
  "scripts": {  
    "start": "node index.js"  
  },  
  "engines": {  
    "node": "^0.12.7"  
  },  
  "dependencies": {  
    "express": "^4.11.0"  
  }  
}
```

Dockerfile

- Now we need a Dockerfile that will build us an OS with nodeJS and installs express using “npm install”
- Dockerfile

```
FROM node:0.12
WORKDIR /app
ADD . /app
RUN npm install
EXPOSE 3000
CMD node index.js
```

Dockerfile

- Now we need to build our Dockerfile

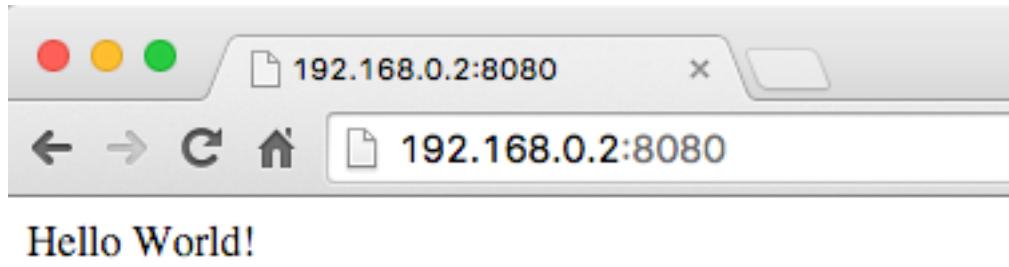
```
vagrant@docker:~$ cd docker-demo
vagrant@docker:~/docker-demo$ docker build .
Sending build context to Docker daemon 15.36 kB
Step 0 : FROM node:0.12
--> 31f630c65071
...
--> 7ea8aef582cc
Successfully built 7ea8aef582cc
```

- Let's run our node app:

```
$ docker run -p 8080:3000 -t 7ea8aef582cc
Example app listening at http://:::3000
```

Dockerfile

- When pointing our browser to port 8080, we get hello world:

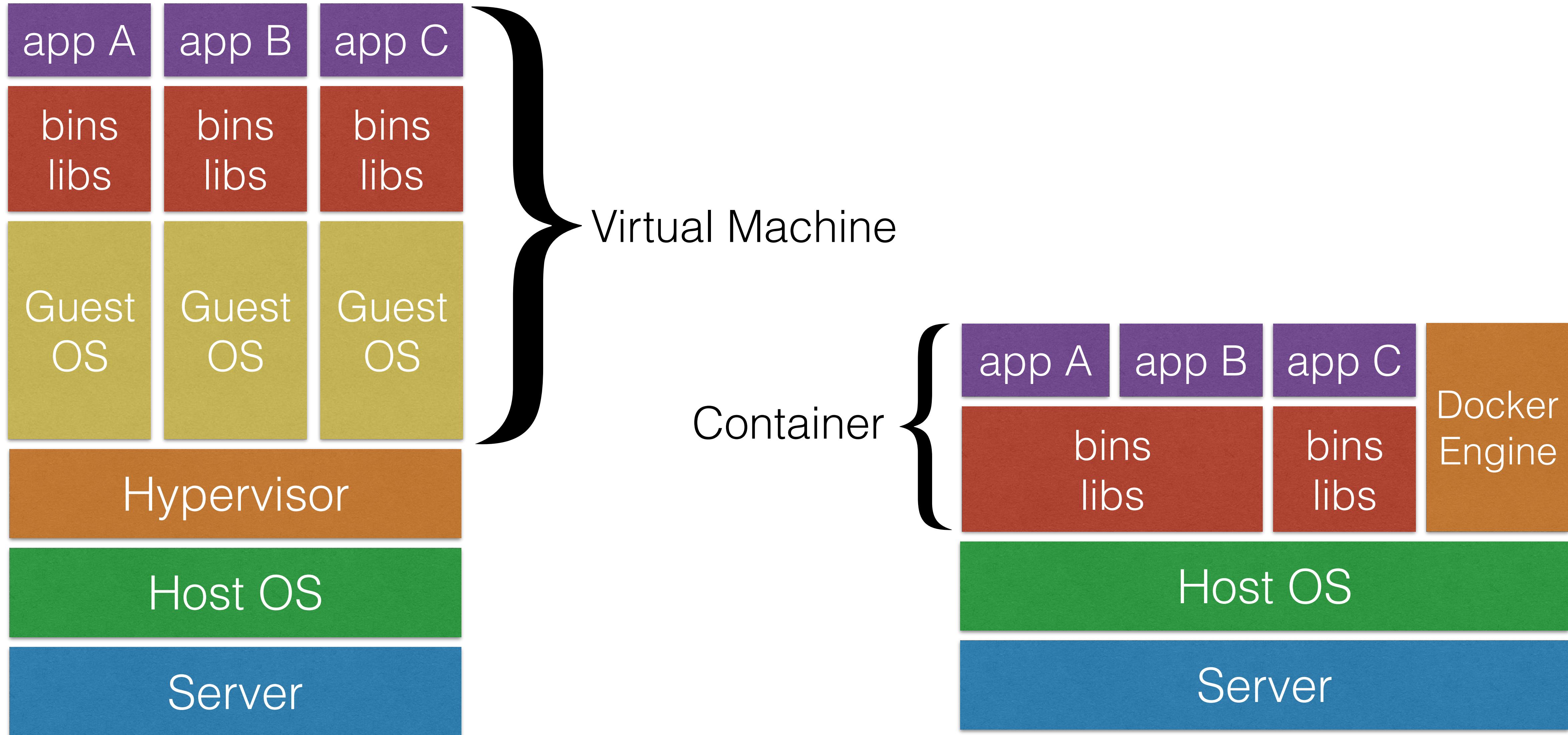


- It works, but it's not very convenient to build images this way
- Better would be to use a container orchestrator to manage containers for us

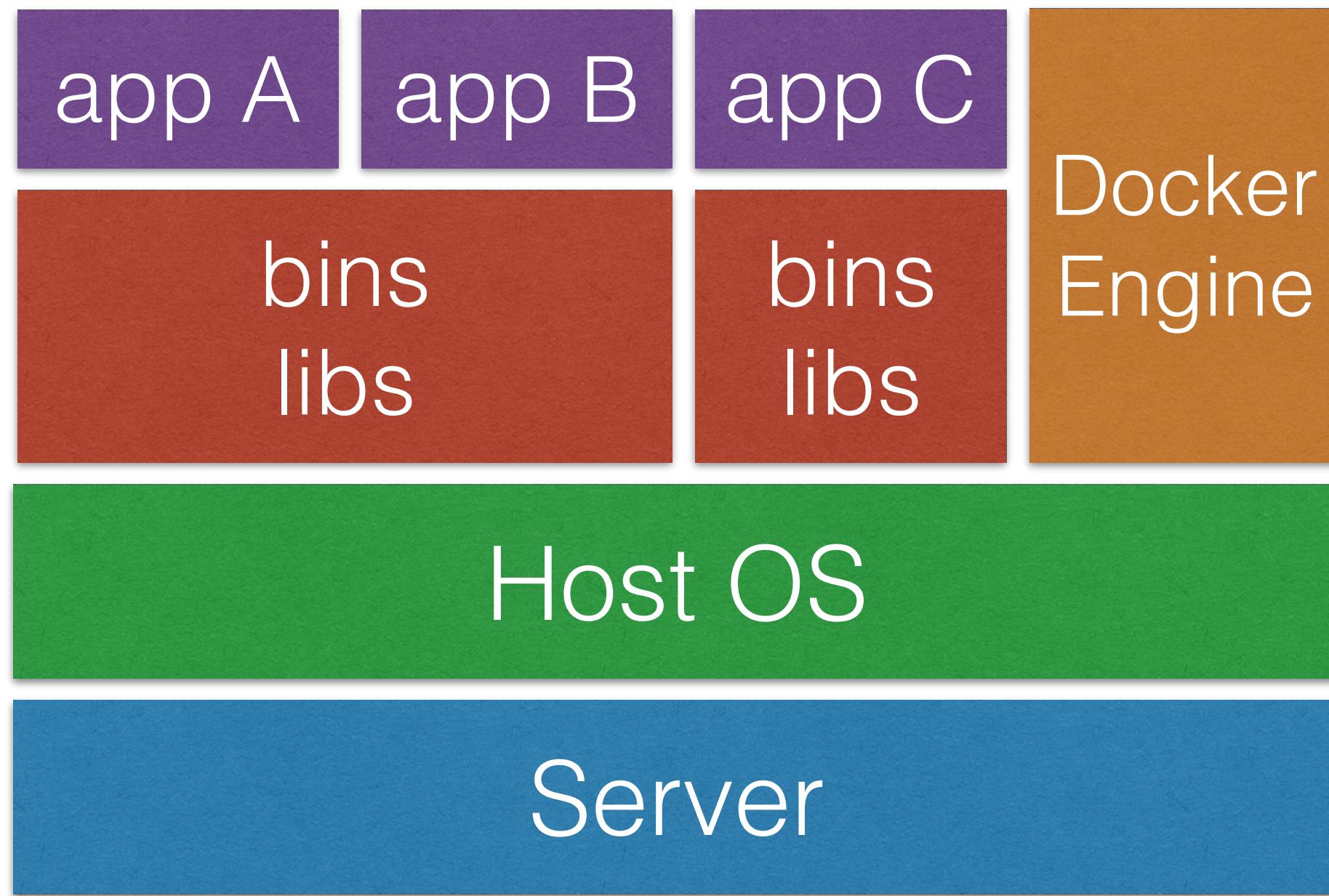
Docker demo

Docker Architecture

Virtual Machines vs Containers



Isolation explained



- Docker (written in the programming language Go) uses functionality in the Host OS to achieve isolation
- Namespaces: Every container has its own pid, net, ipc, mnt, uts namespace
- Control Groups: limitation and prioritization of resources like CPU, Memory, I/O, network, etc.

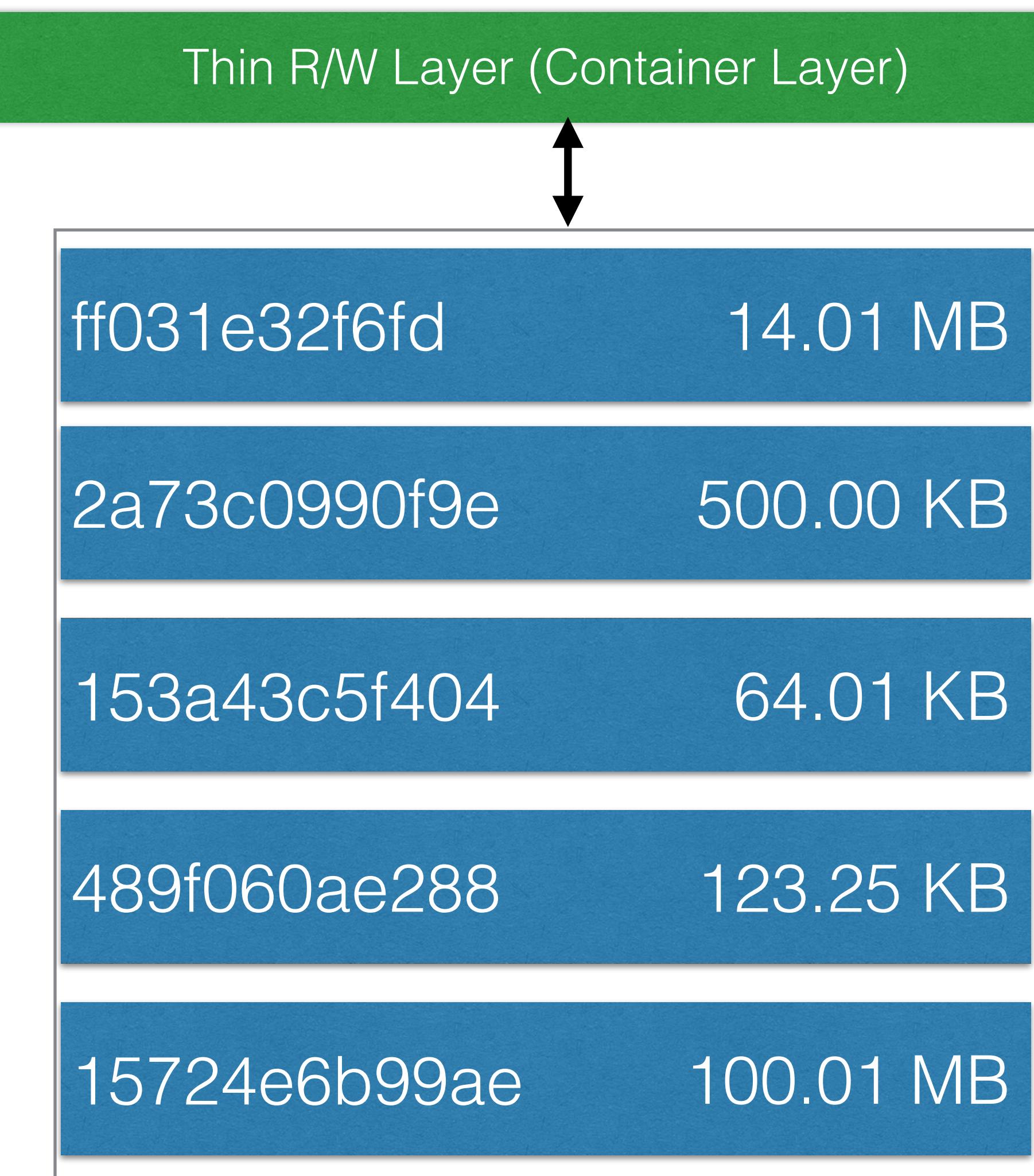
Docker Images and Layers

Ubuntu Image

ff031e32f6fd	14.01 MB
2a73c0990f9e	500.00 KB
153a43c5f404	64.01 KB
489f060ae288	123.25 KB
15724e6b99ae	100.01 MB

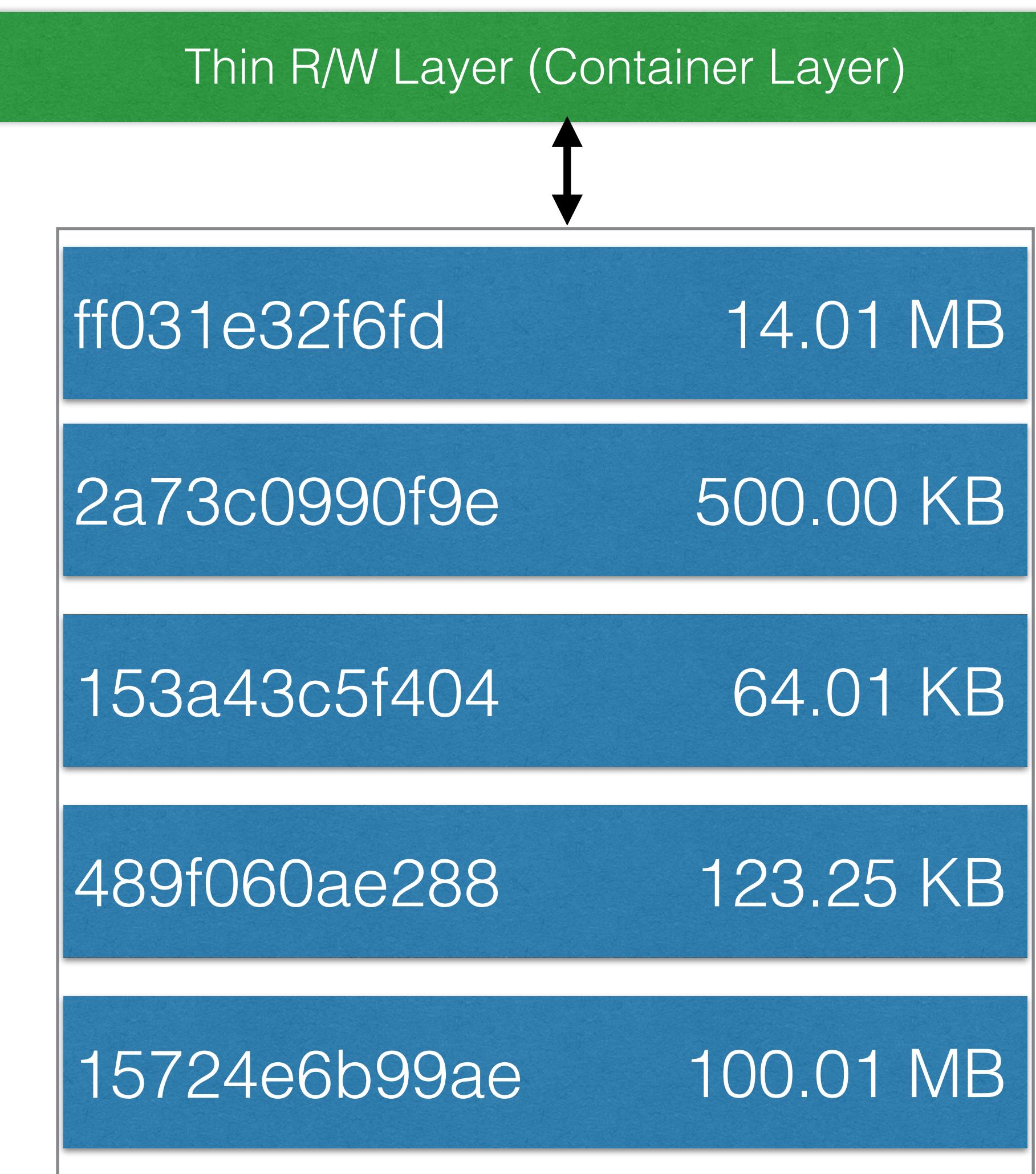
- Docker images are the basis for container, for example the ubuntu image
- Each Docker image references a list of read-only layers
- These layers are stacked together
- Layers can potentially be shared between images
- The Docker storage driver makes the container see these layers as one single image (a single unified view)

Docker Images and Layers



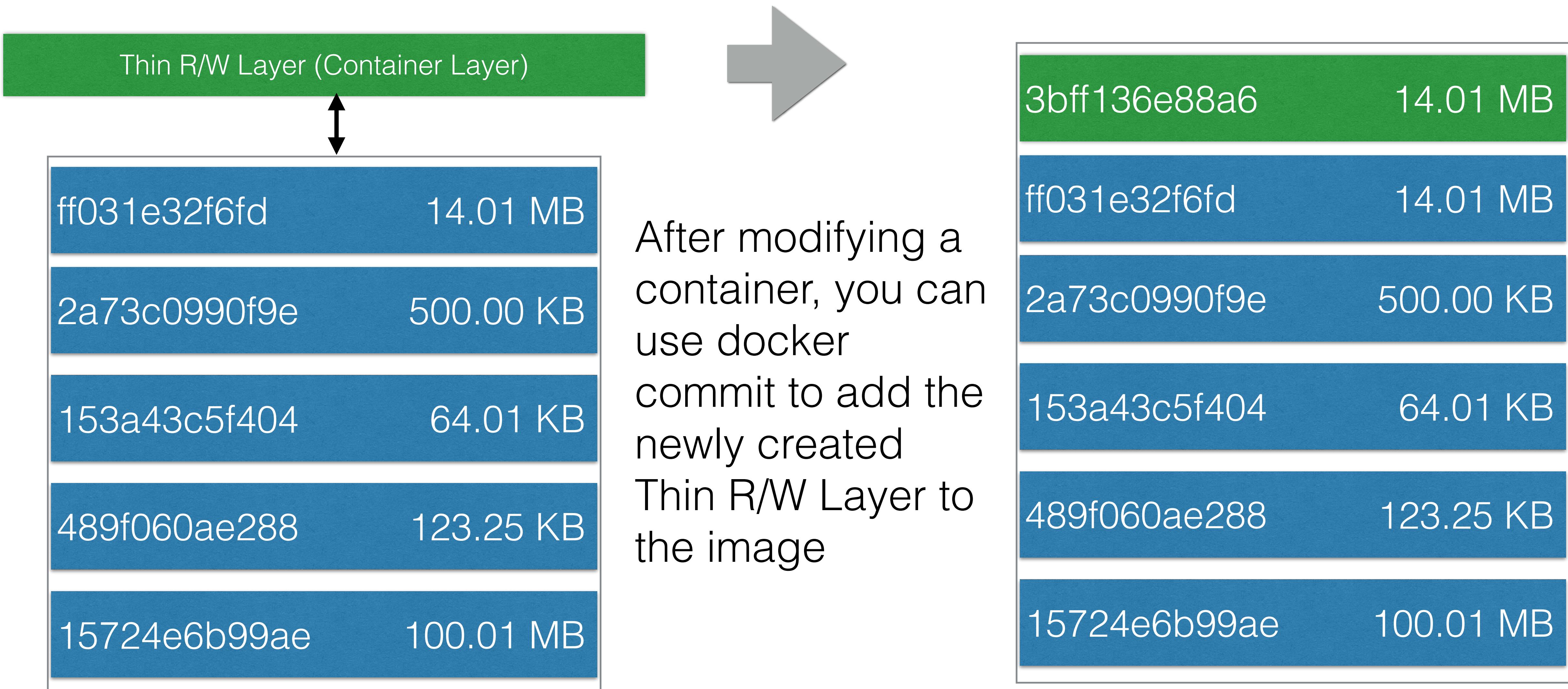
- When you create a new container from an image, a “Thin Read/Write” layer gets added, often called the Container Layer (in green)
- The Read/Write layer is writable by the container (or the user using the container)
- The Image Layers (in blue) are read only

Docker Images and Layers

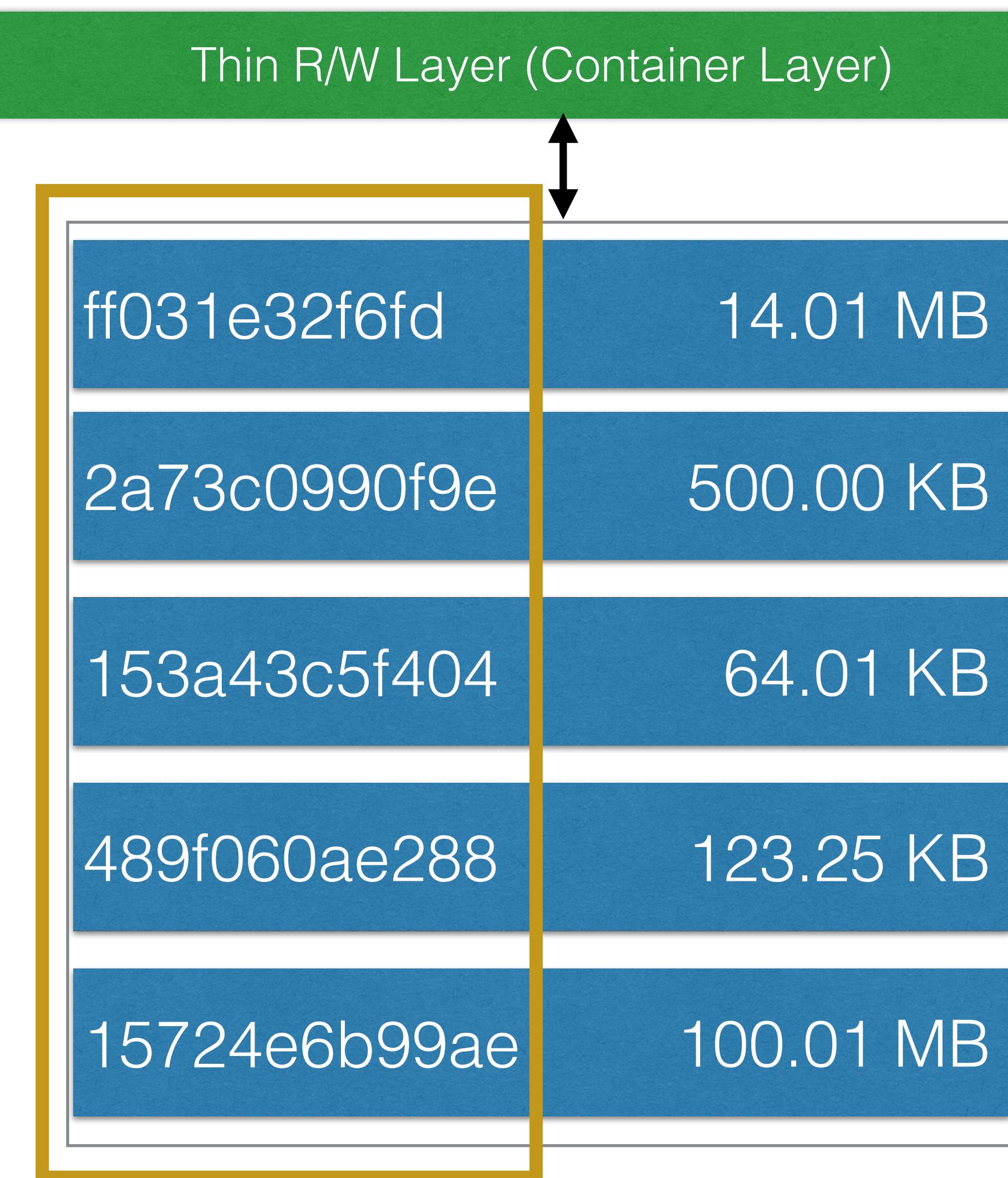


- Every container has its own Container Layer
- Containers can share the readonly image layers and write in their own writable container layer (which is not shared)
- When a container is stopped, all changes made in the Container Layer (in green) are lost
- The underlying image (in blue) layers remain intact when stopping a container

Docker Images and Layers

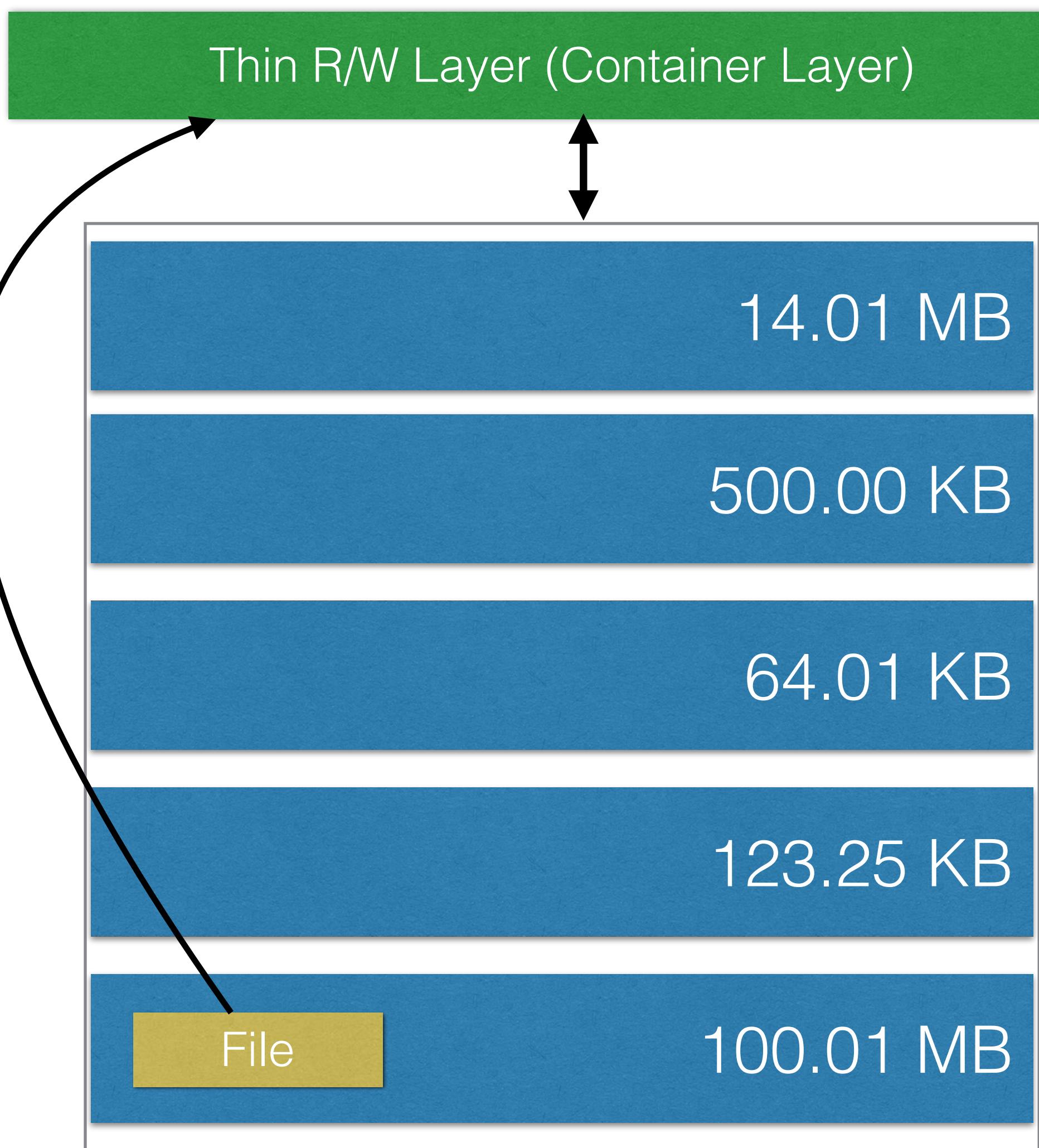


Docker Images and Layers



- Starting from Docker 1.10, layers of images are identified by a secure content hash. Before 1.10, this was done using a randomly generated UUID (Universally Unique Identifier). The Thin R/W Layer still uses UUIDs.
- This new model improves security, avoids ID collisions, guarantees data integrity, and also enables better sharing of layers between images
- In practice you should see better reuse of layers, even if the images didn't come from the same build

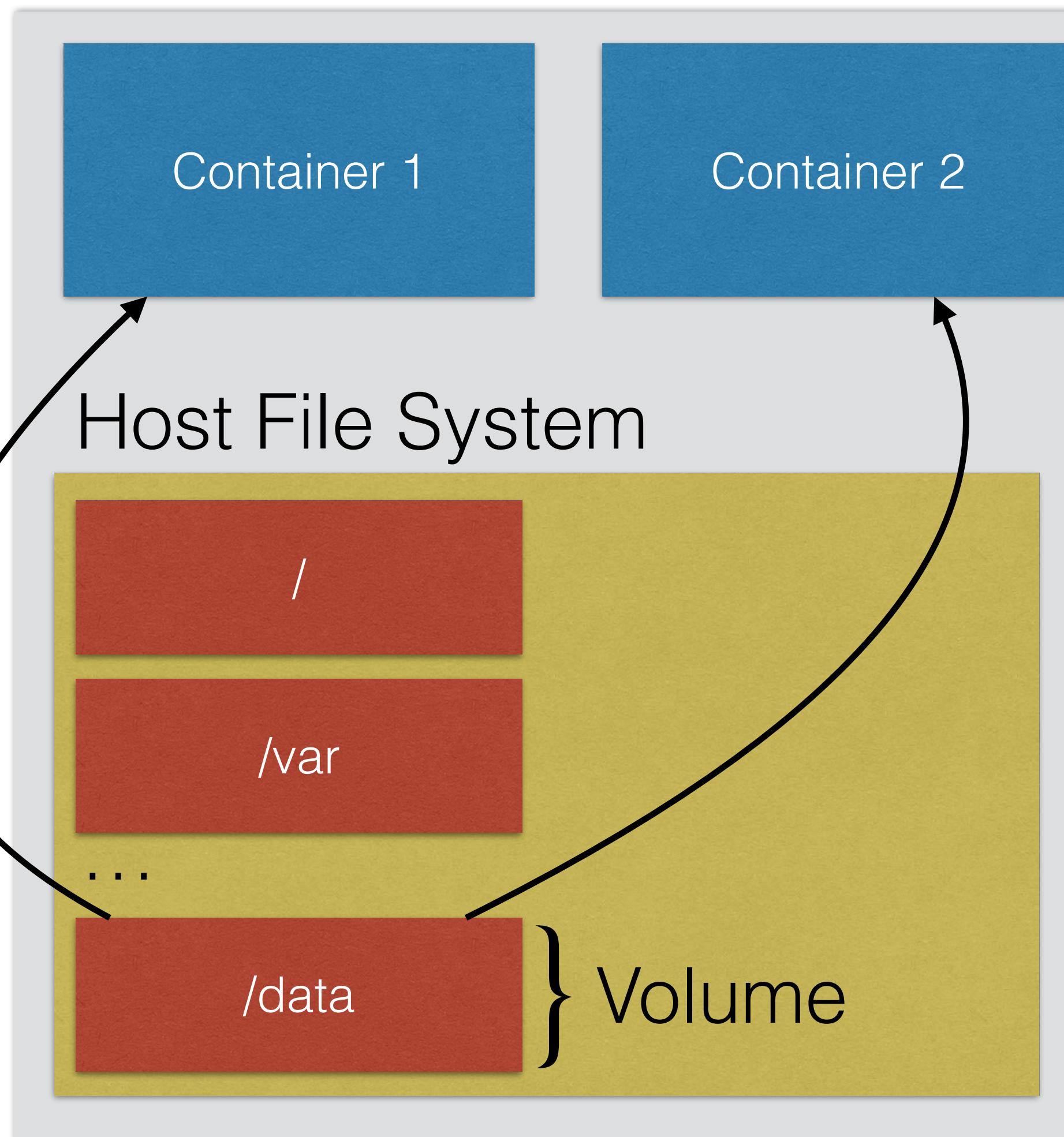
Copy On Write Strategy



- Using the Copy on Write Strategy, processes will share the data as long as both processes have the same data. As soon as one process changes the data, the data will be copied first, then changed.
- Both images as containers use the copy on write strategy to optimize disk space and the performance of a container start
- Multiple containers can safely share the underlying image.
 - When an existing file in the container is modified, a copy on write operation will be performed: the file will be copied “up” from one of the layers to the Thin R/W layer and will then be modified

Docker Volumes

Host OS



- When the container is deleted, the Thin R/W Layer (Container Layer) is removed. Any data stored on this layer is lost
- To persist (keep) data in Docker, you need to use Volumes
- Volumes are files or directories from the Host Operating System that are mounted directly into the container

Docker Volumes

- A volume is a special directory in the container that bypasses the Container Layer (Thin Read-Write Layer) and the read only image layers.
- Volumes are initialized when the container starts
- Data Volumes can be shared by containers
- Changes to the data on the volume will go directly to the host system and will not be saved in the image layers.
- When a container is removed, the data on the Docker Volume remains. It has been written to the Host Operating System (Volume = persistent data store)

Creating a data volume

- To create a data volume, you create a mapping between a container directory and a directory on the host system:

```
vagrant@docker:~$ sudo docker run --name example -v /data/webapp:/webapp --t -i ubuntu:14.04 /bin/bash  
root@af8bae53bdd3:/#
```

- This command launches a Ubuntu LTS container and mounts the directory /data/webapp from the Host Operating System into the container as /webapp
- An application within the container can now use data from the Host Operating System that will be persisted, even when the container is stopped

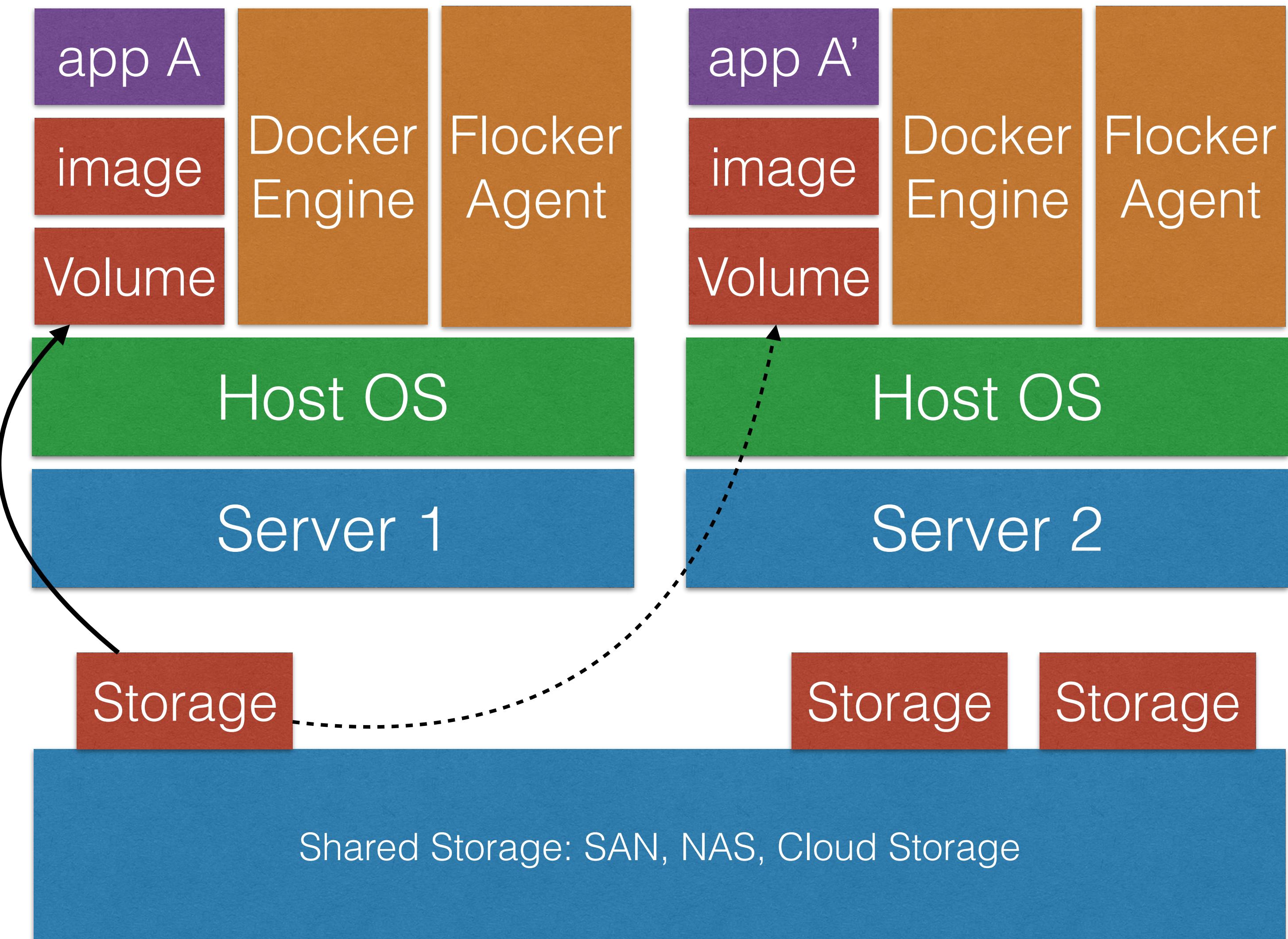
Docker Volume Plugins

- Docker allows you to easily create volumes to persist data on the Host Operating System, where the docker engine runs, but how do you access your data when the Host Operating System is not available anymore?
- Containers can be started on other physical hosts, but how to transfer the data?
 - Stateless applications store their data outside the container and don't experience this problem (see 12-Factor app lectures)
 - Stateful applications need access to their data. We need a system to make sure our data will be available when containers are started on other physical hosts

Docker Volume Plugins

- Docker Volume Plugins enable you to integrate Docker Engine with external storage systems.
- Depending on the storage you will use, the underlying data can be transferred to where the container gets started, independent from the physical host the container is running on
- There are lots of different plugins available. A full list can be found at <https://docs.docker.com/engine/extend/plugins/>
- One promising plugin for stateful applications is Flocker

Docker Volume Plugin: Flocker



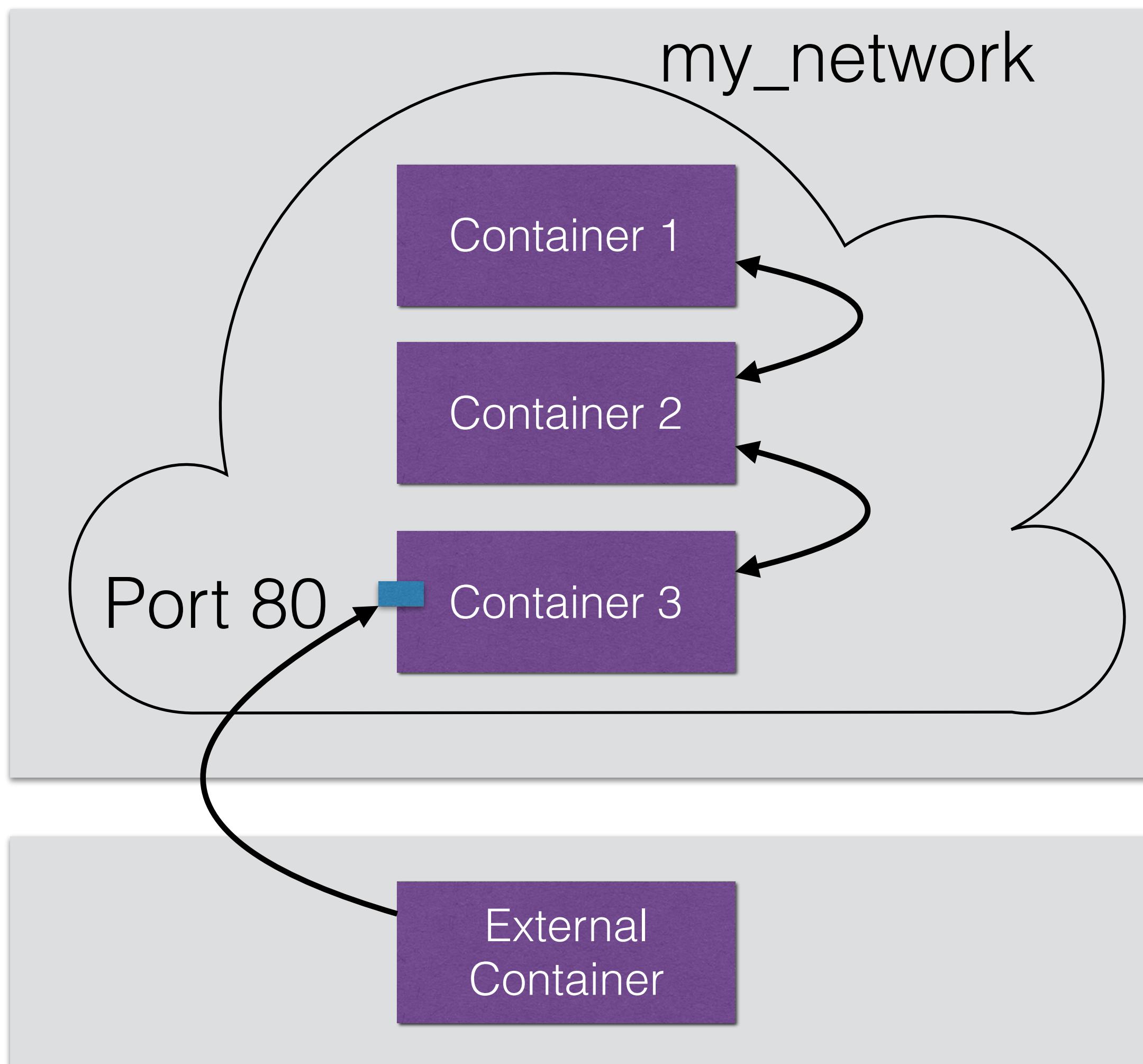
- Flocker plugs into Docker to ensure that Shared Storage can be mapped to the Server where the container runs on
- If a Server crashes, Flocker can remap the storage to another Server, and the container can now be started on the other server, using the same data

Docker Networking

- New Docker containers connect to a bridged network by default
 - The default bridged network allows port mapping and linking between containers, but are difficult to maintain
 - A better way is to create user defined networks
 - Containers within the same user defined network can communicate with each other, but cannot communicate with containers outside the user defined network
 - User defined Bridge Network (on a single Host Operating System)
 - Overlay Network (Multi-Host Operating System)

A User Defined Bridge Network

Docker Host



- Create new Bridge network

```
$ docker network create --driver bridge my_network
```

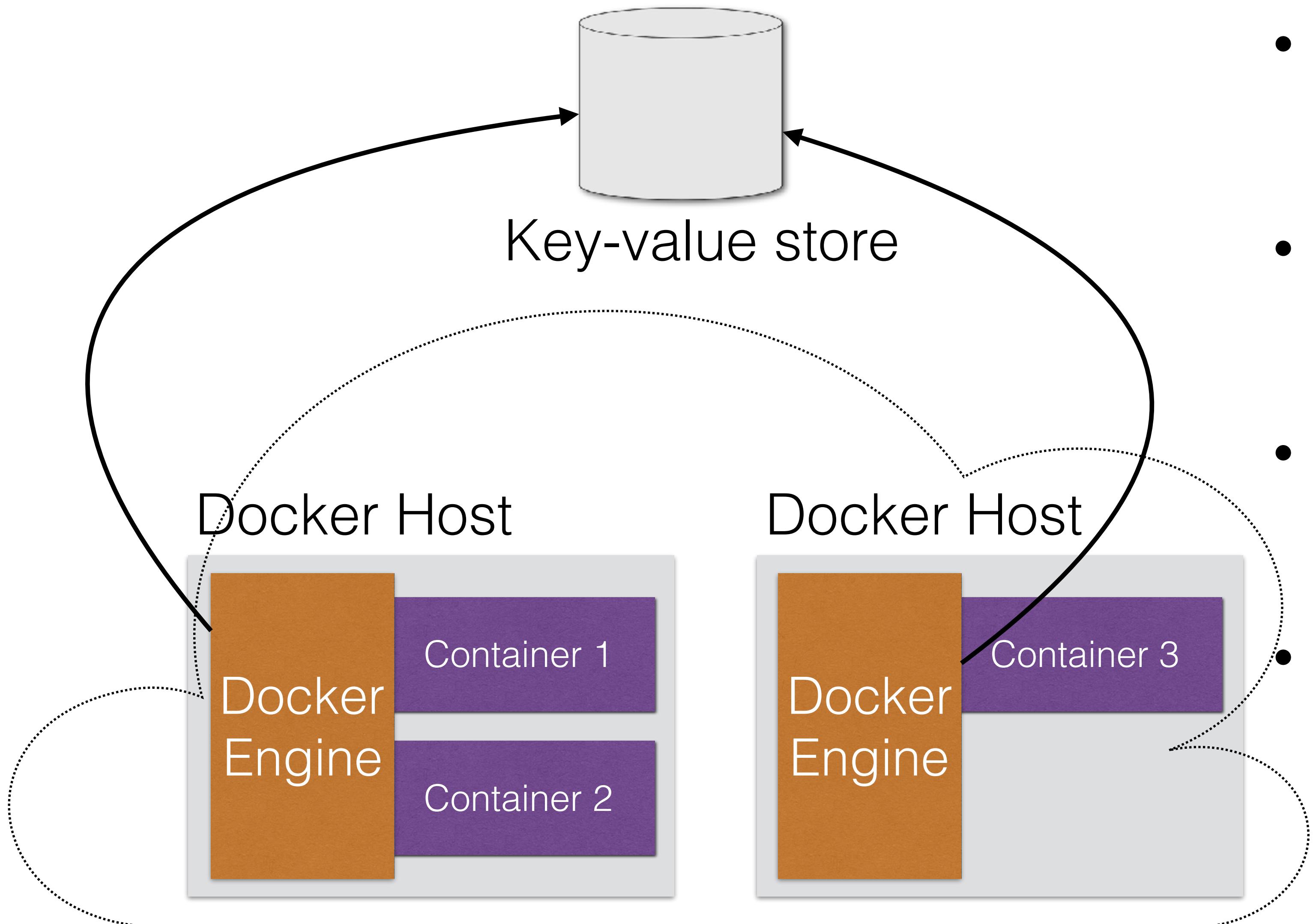
- Launch container in new network

```
$ docker run --net=my_network -itd --name=container1 ubuntu:14.04  
$ docker run --net=my_network -itd --name=container2 ubuntu:14.04
```

- Open port 80 to an external container

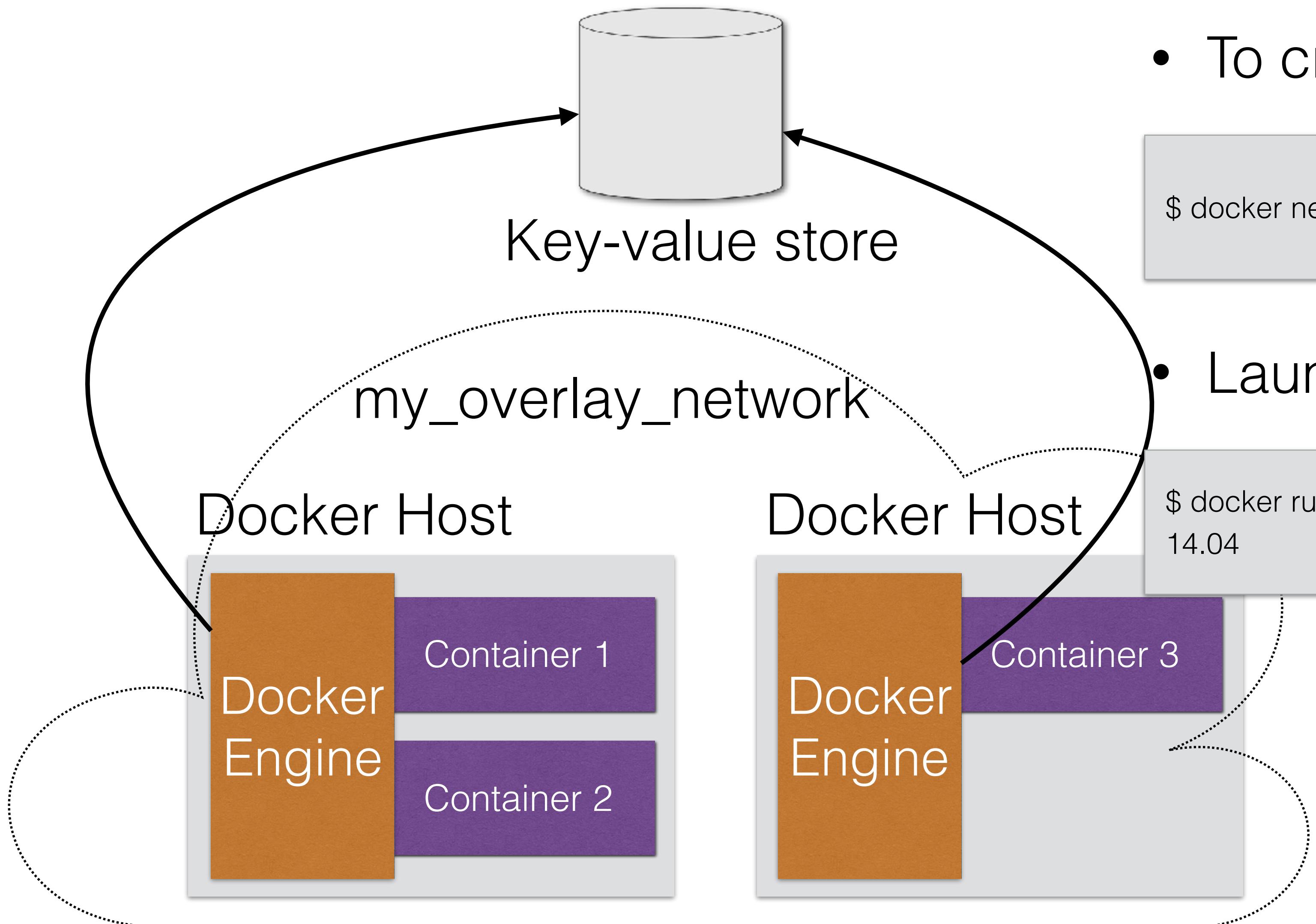
```
$ docker run --net=my_network -itd -p 80:80 --name=container3 ubuntu:  
14.04
```

An Overlay Network



- Docker's overlay network supports multi-hosts networking
- A Key-value store is needed to store the configuration in
- Currently Consul, etcd, and Zookeeper are supported
- You can start your Key-value store on 1 node, but eventually 3 or 5 nodes are recommended

An Overlay Network



- To create an overlay network

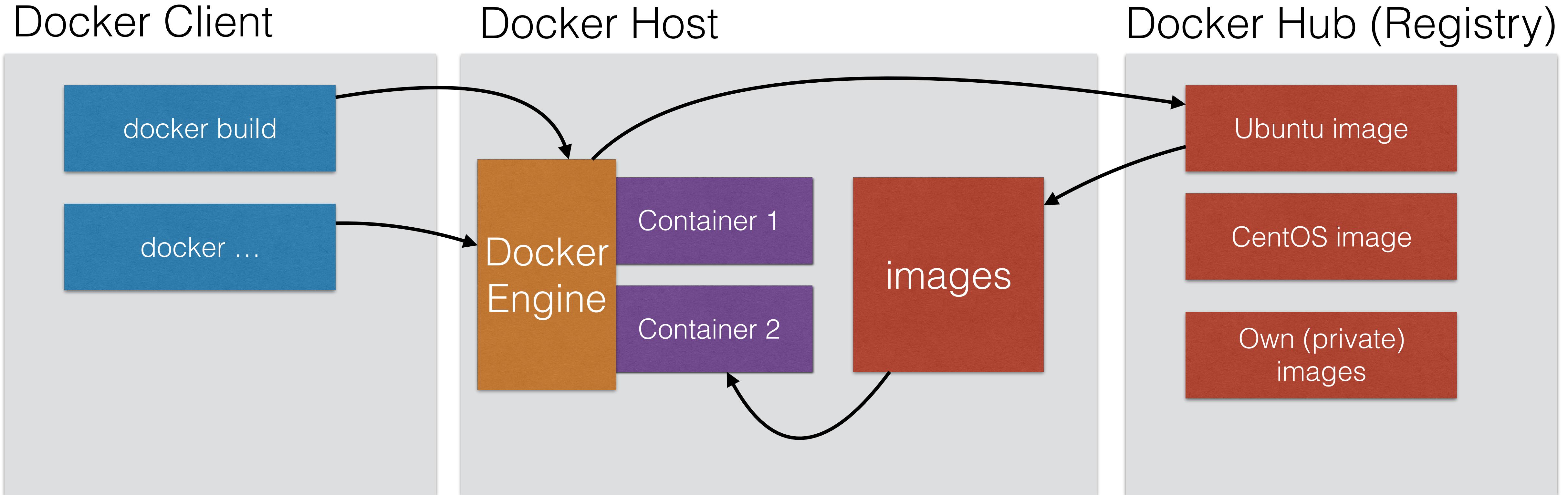
```
$ docker network create --driver overlay my_overlay_network
```

- Launch container in new network

```
$ docker run --net=my_overlay_network -itd --name=container1 ubuntu:14.04
```

Docker Hub

- Docker Hub is a service that provides public and private registries for your Docker Images
- Alternatives are available, e.g. AWS Docker Registry or self-hosted



Docker Compose

Docker Compose

- Compose is a tool part of Docker to define your multi-container docker environment as code
- Compose can manage the lifecycle of your docker containers
 - Typically you write a Dockerfile for every container you want to launch
 - You launch and administrate the containers with the docker-compose command

Docker Compose

- Docker compose is great if you have a multi-container setup
- It can be used during development, testing, or staging environments
- In your Continuous Integration process, it can be used to test an application that depends on external services. The external service can be launched with docker-compose in a container
- We'll use docker compose in later chapters too!

Docker Compose Example

- Let's take another look at our previous nodeJS example. This was our Dockerfile:

```
FROM node:0.12
WORKDIR /app
ADD . /app
RUN npm install
EXPOSE 3000
CMD node index.js
```

- Let's say we would like to use a database backend, and run this database in a separate container
- This is a good use-case for Docker Compose

Docker Compose Example

docker-compose.yml

```
web:  
  build: .  
  command: node index-db.js  
  ports:  
    - "3000:3000"  
  
  links:  
    - db  
  
  environment:  
    MYSQL_DATABASE: myapp  
    MYSQL_USER: myapp  
    MYSQL_PASSWORD: mysecurepass  
    MYSQL_HOST: db  
  
db:  
  image: orchardup/mysql  
  ports:  
    - "3306:3306"  
  
  environment:  
    MYSQL_DATABASE: myapp  
    MYSQL_USER: myapp  
    MYSQL_PASSWORD: mysecurepass
```

- This docker-composer.yml file launches 2 containers: a db and an app container
- Our “web” container is defined on the first line and builds the Dockerfile based in the current path
- Our web container depends on another container, the db container
- The db container is an image pulled from Docker Hub

Docker Compose Example

docker-compose.yml

```
web:  
  build: .  
  command: node index-db.js  
  ports:  
    - "3000:3000"  
  links:  
    - db  
environment:  
  MYSQL_DATABASE: myapp  
  MYSQL_USER: myapp  
  MYSQL_PASSWORD: mysecurepass  
  MYSQL_HOST: db  
  
db:  
  image: orchardup/mysql  
  ports:  
    - "3306:3306"  
environment:  
  MYSQL_DATABASE: myapp  
  MYSQL_USER: myapp  
  MYSQL_PASSWORD: mysecurepass
```

- The environment variables shown here are passed to the app
- The mysql image will create a database and add a user after docker has spun up the image
- The host, database, user, and password are passed to the app as environment variables
- In the next demo I will show this docker-compose.yml in action

Demo

Docker Compose with nodeJS demo

Docker Machine

Docker Machine

- Docker Machine enables you to provision the Docker Engine on virtual hosts
- Once provisioned, you can use Docker Machine to manage those hosts (for example to upgrade the Docker client and daemon)
- You can use Docker Machine to provision Docker hosts on remote systems (on cloud, for example on Amazon AWS)
- You can also use Docker Machine to provision Docker using VirtualBox on your physical Windows / Mac desktop/laptop

Docker Toolbox

- If you are interested in using Docker Machine to run Docker straight from Mac or Windows, without Vagrant in the middle, then take a look at Docker Toolbox:
 - <https://www.docker.com/products/docker-toolbox>
- Docker Toolbox is available for Mac and Windows and comes with Docker Engine, Compose, and Machine
- It also includes Kinematic, a graphical user interface to manage your containers

Provision Docker on Cloud

Docker Client

docker-machine

- Using Docker Machine you can provision instances with Docker on the cloud, for instance on Amazon AWS
- You can run docker-machine directly from your workstation

Docker Host (AWS)

Docker Engine

Container 1
Container 2

Docker Host (AWS)

Docker Engine

Container 3

Demo

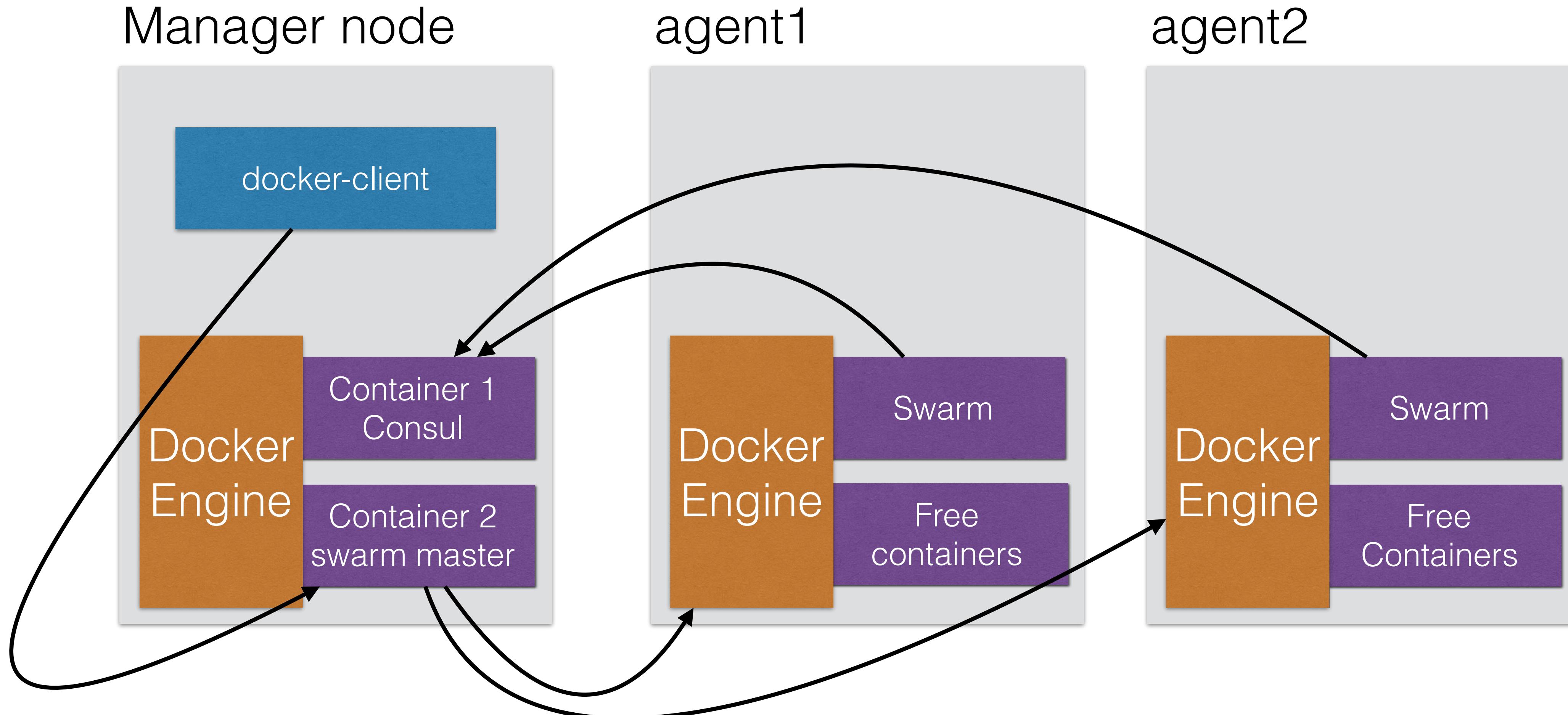
Docker Machine on AWS

Docker Swarm

Docker Swarm

- Docker Swarm enables you to natively cluster Docker containers
- It is built on the standard Docker API. Any tool that communicates with the Docker Daemon, can use Swarm to scale to multiple hosts
- To start using swarm, the Docker Swarm image needs to be pulled and configured
- You can use docker-machine to provision hosts and install swarm
- A discovery backend can also be enabled when using a key-value store (consul, etcd, or zookeeper)

Docker Swarm Architecture



Using Docker Swarm

- To start a swarm cluster, we need a new Vagrantfile to start 3 nodes

```
Vagrant.configure(2) do |config|
  config.vm.define "manager" do |manager|
    manager.vm.box = "ubuntu/trusty64"
    manager.vm.network "private_network", ip: "192.168.0.248"
    manager.vm.hostname = "docker-manager.example.com"
    manager.vm.provision "shell", path: "docker-swarm-demo/docker_install.sh"
  end
  config.vm.define "agent1" do |agent1|
    agent1.vm.box = "ubuntu/trusty64"
    agent1.vm.network "private_network", ip: "192.168.0.247"
    agent1.vm.hostname = "docker-agent1.example.com"
    agent1.vm.provision "shell", path: "docker-swarm-demo/docker_install.sh"
  end
  config.vm.define "agent2" do |agent2|
    agent2.vm.box = "ubuntu/trusty64"
    agent2.vm.network "private_network", ip: "192.168.0.246"
    agent2.vm.hostname = "docker-agent2.example.com"
    agent2.vm.provision "shell", path: "docker-swarm-demo/docker_install.sh"
  end
end
```

- Alternatively, you can install Docker Toolbox on your machine and use docker-machine

Using Docker Swarm

- To have Docker Engine installed automatically on all our hosts, I'm using a shell script, which is referenced in the Vagrantfile
- To get the shell script, git clone my repository first and then start the VMs:

```
$ git clone https://github.com/wardviaene/docker-swarm-demo
$ vagrant up manager
$ vagrant up agent2
$ vagrant up agent3
```

- This is the contents of the docker_install.sh shell script:

```
#!/bin/sh
apt-get update
apt-get -y install apt-transport-https ca-certificates
sudo apt-key adv --keyserver hkp://p80.pool.sks-keyservers.net:80 --recv-keys 58118E89F3A912897C070ADBF76221572C52609D
echo "deb https://apt.dockerproject.org/repo ubuntu-trusty main" >/etc/apt/sources.list.d/docker.list
apt-get update
apt-get -y install linux-image-extra-$(uname -r)
apt-get -y install docker-engine
usermod -aG docker vagrant
```

Using Docker Swarm

- First, you need to install the swarm container on the manager node:

```
vagrant@manager:~$ docker run swarm --help
Unable to find image 'swarm:latest' locally
latest: Pulling from library/swarm
844fab328d6a: Pull complete
d53941759232: Pull complete
3445c6fe19be: Pull complete
a3ed95caeb02: Pull complete
Digest: sha256:51a8eba9502f1f89eef83e10b9f457cf67193efc3edf88b45b1e910dc48c906
Status: Downloaded newer image for swarm:latest
Usage: swarm [OPTIONS] COMMAND [arg...]

A Docker-native clustering system

Version: 1.1.0 (a0fd82b)

Options:
  --debug      debug mode [$DEBUG]
  --log-level, -l "info" Log level (options: debug, info, warn, error, fatal, panic)
```

Using Docker Swarm

- There are multiple ways to run a discovery service for your nodes:
 - You can use a token. The swarm container connects to the Docker Hub discovery service and gets a unique ID, called the discovery token. Using tokens is only suitable for Development and testing.
 - Using a discovery backend service. You can use etcd, Consul, or Zookeeper for this. This is recommended for production
 - In my example, I'll use a single Consul container. This is easy to setup, but not fault tolerant. You should set up at least 3 Consul nodes in production

Using Docker Swarm

- First, start consul on the master

```
vagrant@manager:~$ docker run -d -p 8500:8500 --name=consul program/consul -server -bootstrap  
...  
vagrant@manager:~$
```

- Then, start the swarm manage node on the master

```
vagrant@manager:~$ docker run -d -p 4000:4000 swarm manage -H :4000 --advertise 192.168.0.248:4000 consul://192.168.0.248:8500  
...  
vagrant@manager:~$
```

Using Docker Swarm

- On the 2 agent's, run swarm join to let them join the cluster

```
vagrant@agent1:~$ docker run -d swarm join --advertise=192.168.0.247:2375 consul://192.168.0.248:8500  
...  
vagrant@agent1:~$
```

```
vagrant@agent2:~$ docker run -d swarm join --advertise=192.168.0.246:2375 consul://192.168.0.248:8500  
...  
vagrant@agent2:~$
```

Using Docker Swarm

- You can now check on the master to see the nodes:

```
vagrant@manager:~$ docker -H :4000 info
Containers: 12
Running: 2
Paused: 0
Stopped: 10
Images: 2
Role: primary
Strategy: spread
Filters: health, port, dependency, affinity, constraint
Nodes: 2
docker-agent1: 192.168.0.247:2375
  └ Status: Healthy
...
  └ UpdatedAt: 2016-02-09T11:46:16Z
docker-agent2: 192.168.0.246:2375
  └ Status: Healthy
...
```

Using Docker Swarm

- To run a container on your swarm cluster, use docker run through swarm:

```
vagrant@manager:~$ docker -H :4000 run -p 3000:3000 -d wardviaene/nodejs-demo  
b6c2cd889ecedb0a88d8a5f34f64ef04b7a63a99eedc0d04b4f6bf99d1194352  
vagrant@manager:~$
```

- Now we need to figure out where the container has started:

```
vagrant@manager:~$ docker -H :4000 ps  
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS  
NAMES  
b6c2cd889ece      wardviaene/nodejs-demo   "/bin/sh -c 'node ind'"   25 seconds ago    Up 24 seconds  
192.168.0.247:3000->3000/tcp   docker-agent1/jovial_chandrasekhar
```

- And then we can connect to it:

```
$ curl 192.168.0.247:3000  
Hello World!
```

Docker Swarm and Compose

- Docker swarm and compose are going to get integrated, but it's currently (early 2016) in experimental
- If you are interested in running multiple containers on a Docker Swarm cluster, try the following exercise:
 - Take the nodeJS with MySQL example and try to run it on your swarm cluster
 - Use the continuously evolving documentation on <http://docs.docker.com> to see how
 - Use the discussion boards (or our Facebook group) to get support trying this out!

Demo

Docker Swarm

Docker with Jenkins

Docker and Jenkins

- It's possible to integrate docker in the Continuous Integration flow
- There are plugins available for Jenkins to have a build step where a docker image can be built and sent to a registry
- Currently the jenkins-docker plugins are pretty new and might not work in some cases. An alternative solution is to use “shell” as your build step and execute the docker commands on the shell:

```
ID=$(docker build -t wardviaene/docker-example .)
docker tag $ID wardviaene/docker-example:$VERSION
docker tag $ID wardviaene/docker-example:latest
docker push wardviaene/docker-example
```

Container orchestration

Container orchestration

- Build in:
 - Docker Swarm, and Compose
- Container managers (Orchestrators)
 - Mesos
 - Kubernetes
- Platform as a Service (like heroku)
 - Dokku & Deis
 - Flynn

Kubernetes

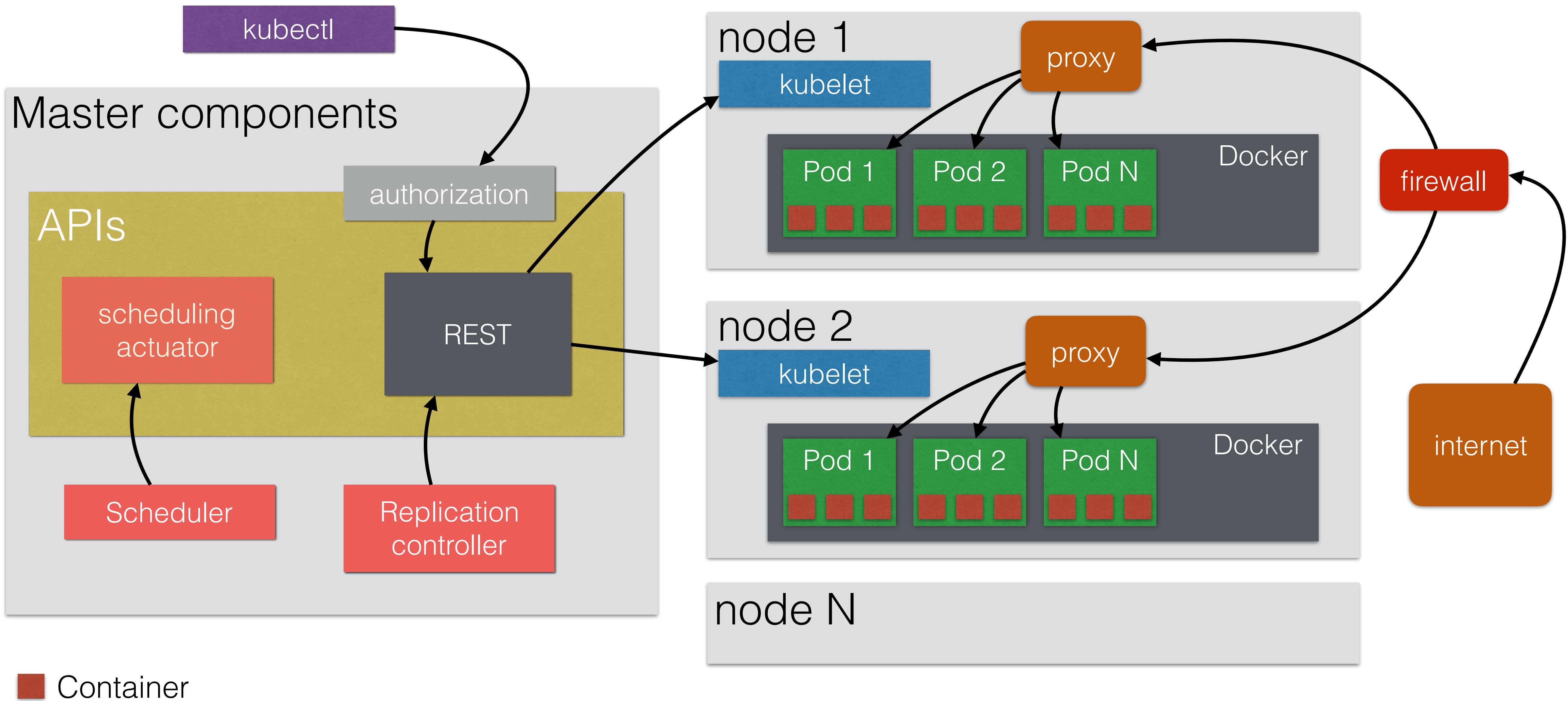
What is kubernetes

- Kubernetes is an open source orchestration system for Docker containers
- It handles scheduling onto nodes in a compute cluster and actively manages workloads to ensure that their state matches the users declared intentions
- Using the concepts of "labels" and "pods", it groups the containers which make up an application into logical units for easy management and discovery

Why Kubernetes

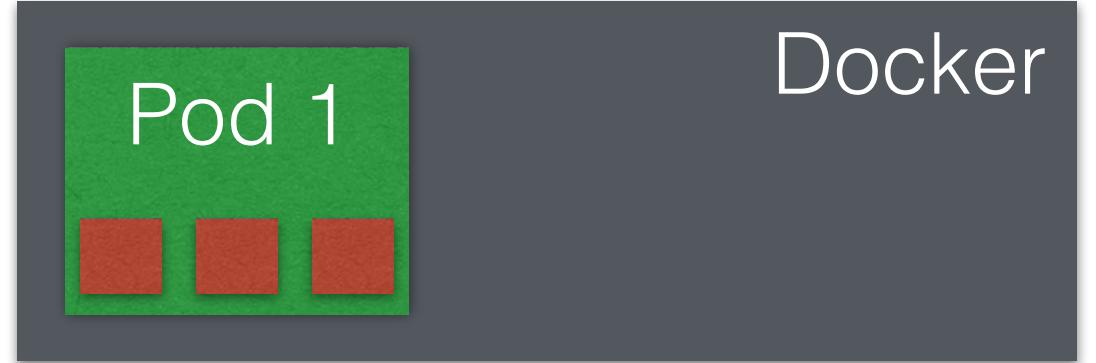
- Lean
 - Lightweight, simple
- Portable
 - Public (Google cloud), private, or hybrid cloud
- Extensible
 - Highly modular, designed so that all of its components are easily swappable
- Open source

Architecture overview



Kubernetes pod

- A pod consists of a colocated group of Docker containers with shared volumes
- Pods are the smallest deployable units that can be created, scheduled, and managed.
- Kubernetes Pods can come and go, if one of them shuts down or crashes, a new pod will be started
- When scaling up or down or when doing rolling updates pods are created or destroyed.
- This leads to a problem: if some set of Pods (e.g. backends) provides functionality to other Pods (e.g. frontends), how do those frontends find out and keep track of which backends are in that set

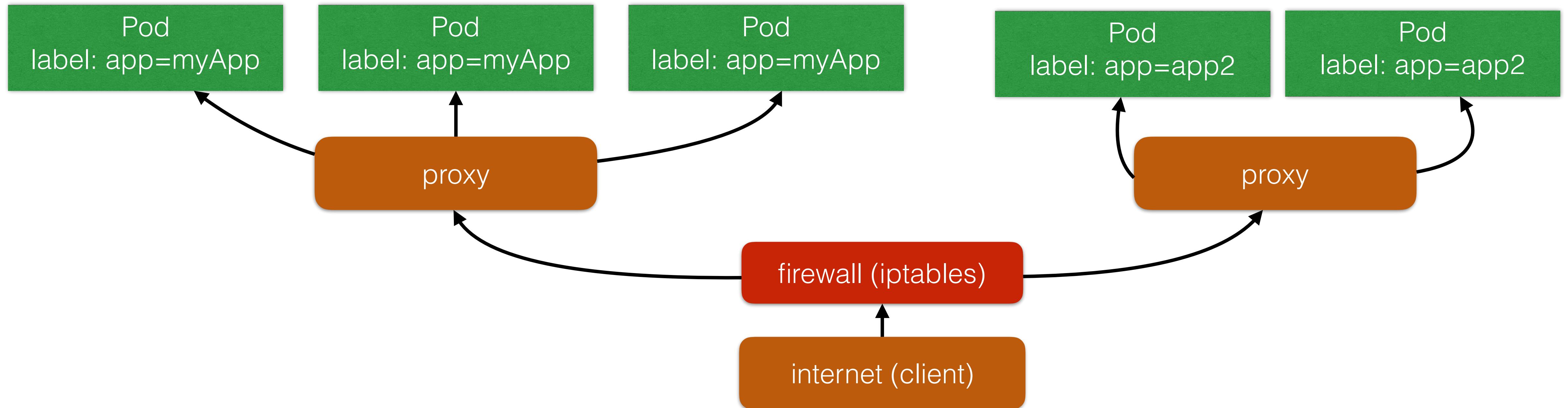


Kubernetes Service

- A Kubernetes Service is an abstraction which defines a logical set of Pods and a policy by which to access them - sometimes called a micro-service.
- Any traffic bound for the Service is proxied to an appropriate backend without the clients knowing anything about Kubernetes or Services or Pods.
- Unlike Pod IP addresses, which actually route to a fixed destination, Service IPs are not actually answered by a single host. Instead, kubernetes use iptables (packet processing logic in Linux) to define virtual IP addresses which are transparently redirected as needed.

Kubernetes Service

- When a client connects to the virtual IP of the firewall, an iptables rule kicks in, and redirects the packets to the proxy. The proxy selects a pod based on the label, and starts proxying traffic from the client to the backend.



Setting up a new machine

Vagrantfile

```
Vagrant.configure(2) do |config|
  config.vm.define "k8s" do |k8s|
    k8s.vm.box = "ubuntu/trusty64"
    k8s.vm.network "private_network", ip: "192.168.0.251"
    k8s.vm.hostname = "k8s.example.com"
    k8s.vm.provider "virtualbox" do |v|
      v.memory = 1024
    end
  end
end
```

Starting machine

```
$ vagrant up k8s
$ vagrant ssh k8s
```

Docker setup

- Install Docker

```
vagrant@k8s:~$ sudo apt-get install docker.io
```

- Enable cgroups and swap accounting, edit /etc/default/grub.conf:

```
GRUB_CMDLINE_LINUX="cgroup_enable=memory swapaccount=1"
```

- Then reboot

```
vagrant@k8s:~$ sudo update-grub  
vagrant@k8s:~$ sudo reboot  
vagrant@k8s:~$
```

Start kubernetes

- Step1: Run etcd

```
vagrant@k8s:~$ sudo docker run —volume=/var/etcd:/var/etcd --net=host -d gcr.io/google_containers/etcd:2.0.12 /usr/local/bin/etcd --addr=127.0.0.1:4001 --bind-addr=0.0.0.0:4001 --data-dir=/var/etcd/data
```

- Step2: run the master

```
vagrant@k8s:~$ sudo docker run \  
--volume=/:/rootfs:ro \  
--volume=/sys:/sys:ro \  
--volume=/dev:/dev \  
--volume=/var/lib/docker:/var/lib/docker:ro \  
--volume=/var/lib/kubelet:/var/lib/kubelet:rw \  
--volume=/var/run:/var/run:rw \  
--net=host \  
--pid=host \  
--privileged=true \  
-d \  
gcr.io/google_containers/hyperkube:v1.0.1 \  
/hyperkube kubelet --containerized --hostname-override="127.0.0.1" --address="0.0.0.0" --api-servers=http://localhost:8080 --config=/etc/kubernetes/manifests
```

Start kubernetes

- Step Three: Run the service proxy

```
vagrant@k8s:~$ sudo docker run -d --net=host --privileged gcr.io/google_containers/hyperkube:v1.0.1 /hyperkube proxy --master=http://127.0.0.1:8080 --v=2
```

- Download kubectl

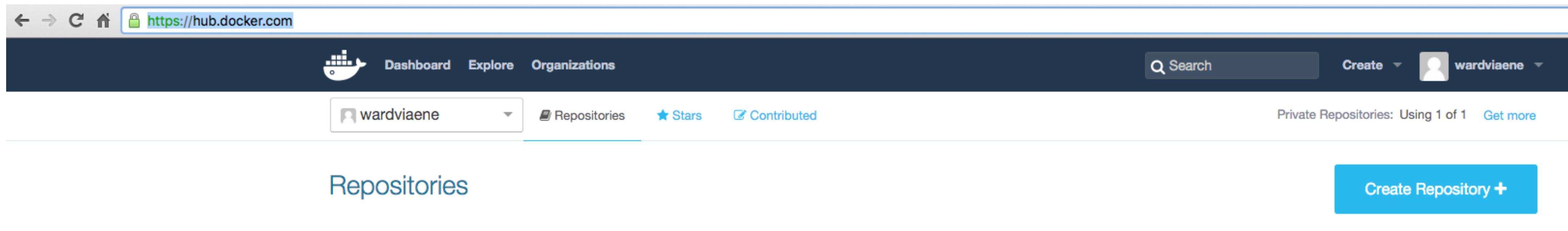
```
vagrant@k8s:~$ wget https://storage.googleapis.com/kubernetes-release/release/v1.0.1/bin/linux/amd64/kubectl  
vagrant@k8s:~$ chmod +x kubectl  
vagrant@k8s:~$ mkdir bin  
vagrant@k8s:~$ mv kubectl bin  
vagrant@k8s:~$ . .profile
```

Creating a pod

- A pod definition is a declaration of a desired state.
- Let's run our nodeJS hello world example app on kubernetes
- We need a pod definition (in yaml) and a docker image in a repository
 - The easiest way is to push your image to Docker Hub
 - You will need to create an account at <https://hub.docker.com>
 - If you don't want to create an account, you can use the docker image that I created for this course: wardviaene/k8s-demo

Creating a pod

- Once you created an account, you can create a new repository



Creating a pod

- You can call it k8s-demo

Create Repository

1. Choose a namespace *(Required)*
2. Choose a name *(Required)*
3. Add a short description
4. Add markdown to the full description field
5. Set it to be a private or public repository

The screenshot shows a 'Create Repository' interface. At the top, there are two dropdown menus: 'Namespace' set to 'wardviaene' and 'Name' set to 'k8s-demo'. Below these are two text input fields: 'Short Description' containing 'demo' and 'Full Description' which is currently empty. Underneath these is a 'Visibility' dropdown set to 'public'. At the bottom is a large blue 'Create' button.

Push image

- Docker login

```
vagrant@k8s:~$ sudo docker login
Username: your-docker-hub-username
Password:
Email: your-docker-hub-email
WARNING: login credentials saved in /home/vagrant/.dockercfg.
Login Succeeded
vagrant@k8s:~$
```

- get nodeJS app from git, build, and push

```
vagrant@k8s:~$ git clone https://github.com/wardviaene/docker-demo
vagrant@k8s:~$ cd docker-demo/
vagrant@k8s:~/docker-demo$ sudo docker build -t your-docker-hub-username/k8s-demo .
vagrant@k8s:~/docker-demo$ sudo docker push your-docker-hub-username/k8s-demo
...
vagrant@k8s:~/docker-demo$
```

Create a pod

- Create a file pod-k8s-demo.yml with the pod definition:

```
apiVersion: v1
kind: Pod
metadata:
  name: nodehelloworld.example.com
  labels:
    app: helloworld
spec:
  containers:
  - name: k8s-demo
    image: wardviaene/k8s-demo
    ports:
    - containerPort: 3000
```

- Use kubectl to create the pod on the kubernetes cluster:

```
vagrant@k8s:~$ kubectl create -f k8s-demo/pod-k8s-demo.yml
vagrant@k8s:~$
```

Create a service

- Create a file service-k8s-demo.yml with the service definition:

```
apiVersion: v1
kind: Service
metadata:
  name: helloworld-service
spec:
  ports:
    - port: 3000
      targetPort: 3000
      protocol: TCP
  selector:
    app: helloworld
```

- Use kubectl to create the pod on the kubernetes cluster:

```
vagrant@k8s:~$ kubectl create -f k8s-demo/service-k8s-demo.yml
vagrant@k8s:~$
```

Test the app

- Test the app

```
vagrant@k8s:~$ export SERVICE_IP=$(kubectl get service helloworld-service -o=template -t={{.spec.clusterIP}})  
vagrant@k8s:~$ export SERVICE_PORT=$(kubectl get service helloworld-service -o=template '-t={{(index .spec.ports 0).port}}')  
vagrant@k8s:~$ curl http://${SERVICE_IP}:${SERVICE_PORT}  
hello world!
```

Run multiple pods

- To run multiple instances of your application, use the replication controller:

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: helloworld-controller
spec:
  replicas: 2
  selector:
    app: helloworld
  template:
    metadata:
      labels:
        app: helloworld
    spec:
      containers:
        - name: k8s-demo
          image: wardviaene/k8s-demo
      ports:
        - containerPort: 3000
```

Kubernetes demo

DevOps Challenge

labs!

The DevOps Challenge

- You can find the **DevOps Challenge** at www.devopschallenge.co
- There is **no need to register**, but you can enter your email to save your progress (which is recommended)
- Currently the DevOps Challenge has **multiple levels** covering the following **technologies**:
 - **SSH Keys** creation
 - **Terraform** infrastructure setup (with instructional videos how to do so)
 - **Ansible** challenges (both for Redhat and Debian based systems)

The DevOps Challenge!

This is the first of many challenges, where you take the seat of a starting engineer in a DevOps role. This challenge should teach you in a fun way how to apply DevOps techniques.

[Start Challenge](#)

DevopsChallenge.co

DevOps Challenge

[Home](#) [Login](#) [Contact](#)

Challenge I

You are hired!

You are now an employee Devz Incorporated

It's your **first day** at Devz Incorporated. You are hired as a **DevOps Professional**. You will **automate everything**, increase deployments, decrease holding time in the SDLC, and **prosper a better relationship** with the developers of Devz Incorporated. Expectations are very high, as you are the only person having an understanding of DevOps ... or haven't you?

Hint

No hints to show

[Next!](#)

The DevOps Challenge

Good luck & Enjoy

www.devopschallenge.co

Addendum

Sources

<http://www.slideshare.net/realgenekim/2014-state-of-devops-findings-velocity-conference>

<http://www.scriptrock.com/automation-enterprise-devops-doing-it-wrong>

<http://www.mobify.com/blog/devops-101-best-practices/>

<http://www.slideshare.net/sanjeev-sharma/campdevops-keynote-devops-using-lean-to-eliminate-bottlenecks>

<http://www.slideshare.net/pritiman/intro-to-devops-14053761>

http://www.slideshare.net/geekle/devops-5348895?next_slideshow=1

<http://en.wikipedia.org/wiki/DevOps>

<http://12factor.net>

Further readings

- CD: <http://continuousdelivery.com/>
- GIT: <https://www.atlassian.com/git/tutorials/comparing-workflows/forking-workflow> (CC License)
- git flow: <http://danielkummer.github.io/git-flow-cheatsheet/>
- kubernetes setup: <https://github.com/kubernetes/kubernetes/blob/release-1.1/docs/getting-started-guides/docker.md>

Thank you

