Managing Upgrades with Ansible

Presented by Travis Holton

Managing Upgrades with Ansible

Introduction

About me

- I work at Catalyst here in Wellington
- I write code in
  - Python
  - Perl (long ago)
- Nowadays almost exclusively "devops" related stuff
  - mostly Ansible
  - some Docker
- I also do training at Catalyst
  - Ansible
  - Kubernetes

Where to find me

- GitHub: github.com/heytrav
- Twitter: @heytrav
- Email: travis@catalyst.net.nz

About this course

- Assumes some experience with Ansible
- Assumes knowledge of basic concepts:
  - playbooks
  - plays
  - tasks
  - variables
- YAML

Goals of this workshop

- Examine different types of cluster deployment
- See how we can use built-in features of Ansible to manage deploying and upgrading
- Provide tools that can be used in any build or CI/CD pipeline

Course Outline

- Setup
- Cloud Signup
- Review basics
- Provisioning Machines
- Deploying the Application
- Upgrade Strategies
- In-place rolling upgrade
- Blue Green
- Closing

Source material

- Keating, Jesse. *Mastering Ansible*. Packt, 2015
- Hochstein, Lorin et al. *Ansible Up & Running 2nd Edition*. O'Reilly, 2017
- https://docs.ansible.com/ansible/latest/user_guide/playbooks_delegation.html
- Based somewhat on my own experience

Setup

Checkout the code

- Clone the course material and sample code

```
git clone https://github.com/heytrav/ansible-workshop-2019.git
```

- or follow along on GitHub
  - README -> Course Outline
- ..or follow along in pdf in base directory

Installing Ansible

- For this workshop we'll be using Ansible ≥ 2.8
- Installation options

Setup Python virtualenv

- Requirements
  - ≥ python3.5
  - virtualenv
- Set up local Python environment

```
virtualenv -p `which python3` venv
```

- Activate virtualenv as base of Python interpreter

```
source ~/venv/bin/activate
```

Install Dependencies

- Update Python package manager (pip)

```
pip install -U pip
```

- Install dependencies

```
pip install -r requirements.txt
```

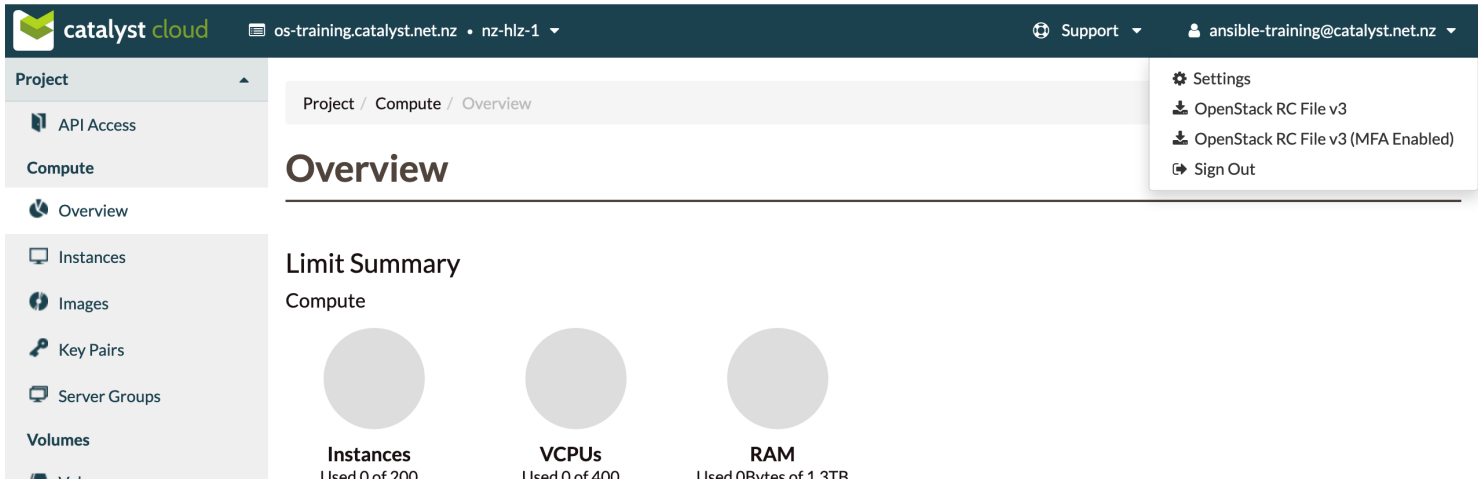
Cloud Provider Account

Catalyst Cloud Registration

- Open https://dashboard.cloud.catalyst.net.nz
- Sign up for an account
- Enter promo code
  - pycon2019

Download OpenStack credentials

- In upper right corner under your account name



- Download *OpenStack RC File v3*

OpenStack SDK login

- Your terminal will need to be logged in to interact with OpenStack on the Catalyst Cloud
- Activate the OpenStack config in your terminal
- This will prompt you to enter your Catalyst Cloud password

```
source <your account name>.catalyst.net.nz-openrc.sh
Please enter your OpenStack Password for project ...
*******
```

Ansible Basics

(quick review)

Terminology

Task

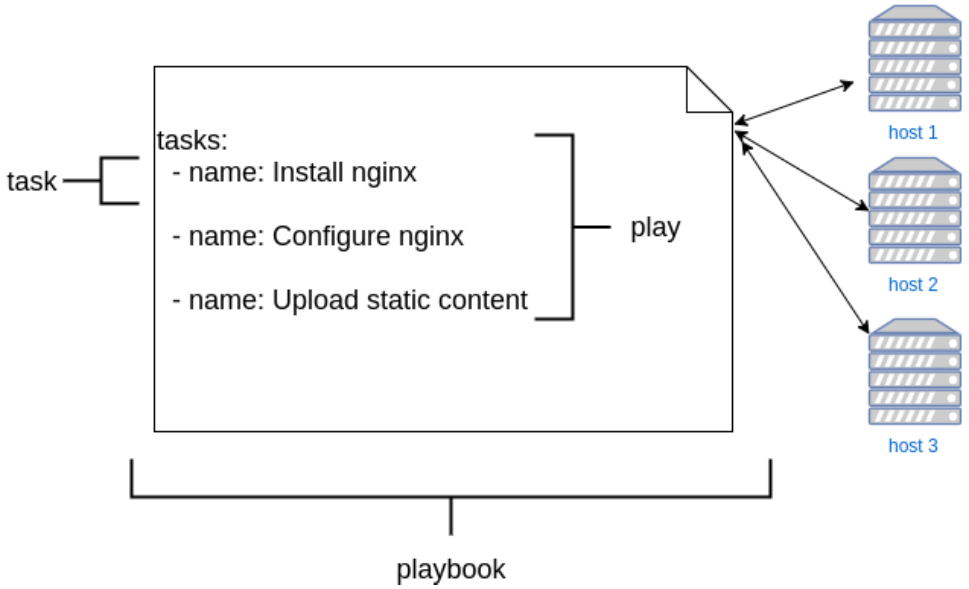
An action to perform

Play

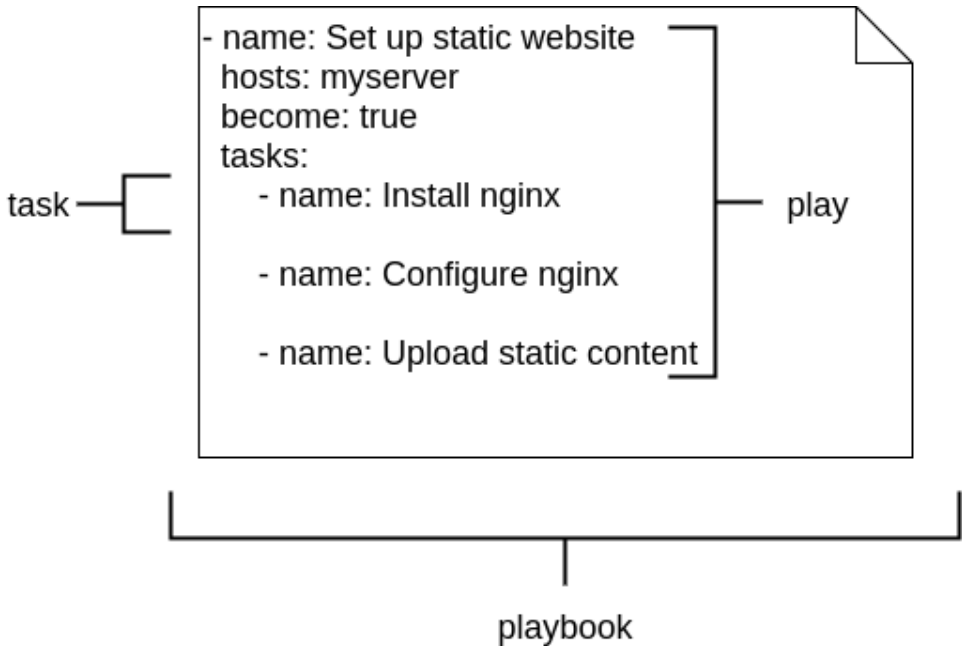
a collection of tasks

Playbook

YAML file containing one or more plays



Ansible Playbook Structure



- A playbook is a YAML file containing a list of plays
- A play is a dictionary object

Ansible Playbook Structure

- A play must contain:
  - `hosts`
    - A string representing a particular host or *group* of hosts
      - `hosts: localhost`
      - `hosts: app.mywebsite.com`
      - `hosts: appserver`
    - These are what you will configure

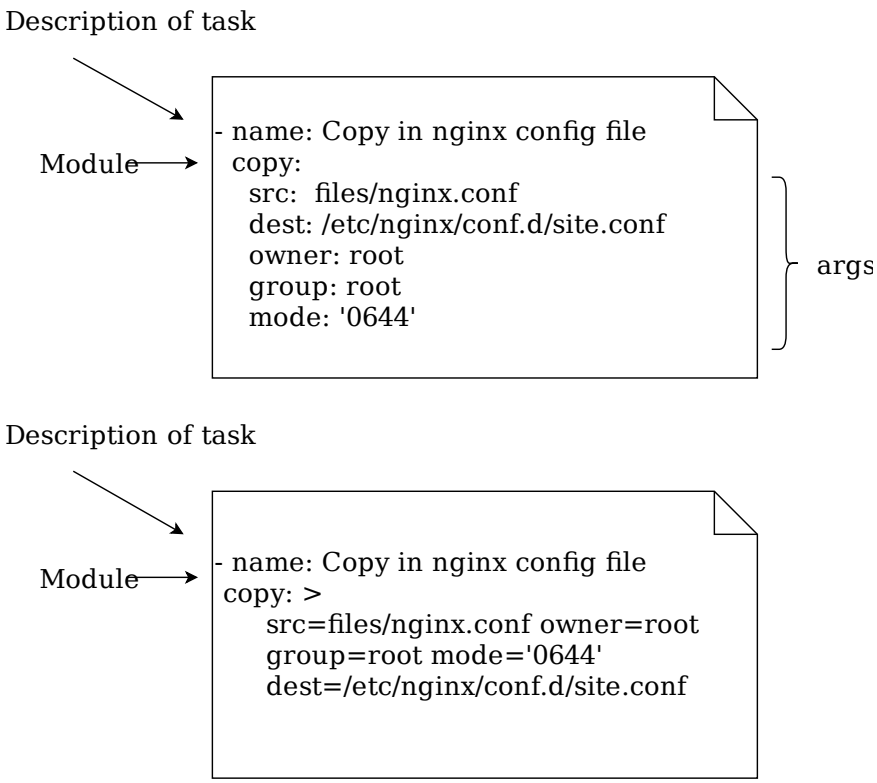
Ansible Playbook Structure

- A play may optionally contain:
  - tasks
    - A list of dictionaries
    - What you want to do
  - name
    - Description of the play
  - vars
    - Variables scoped to the play

Structure of a Task

- A task is a dictionary object containing
  - name
    - Describes what the task does
    - Optional but best practice to use
  - module
    - Dictionary object
    - Key represents Python module which will perform tasks
    - May have arguments

Structure of a Task



- Two styles of module object in tasks
  - string form
  - dictionary form
- Dictionary form is more suitable for complex arguments
- Matter of preference/style

More Terminology

Module

Blob of Python code which is executed to perform task

Inventory

File containing hosts and groups of hosts to run tasks

YAML and indentation

- Ansible is fussy about indentation
- TABS not allowed (Ansible will complain)
- Playbook indentation

```
- name: This is a play
  hosts: somehosts              # 2 spaces
  tasks:
    - name: This is a task      # 4 spaces
      module:                   # 6 spaces
        attr: someattr          # 8 spaces
        attr1: someattr
        attr2: someattr
```

- Task file indentation

```
- name: This is a task
  module: somehosts              # 2 spaces
    attr1: value1                # 4 spaces
    attr2: value2                # 4 spaces
    attr3: value3                # 4 spaces
  loop: "{{ list_of_items }}"    # 2 spaces
```

- Use an editor that supports YAML syntax

Vim YAML setup

- `.vimrc` file

```
syntax on
filetype plugin indent on
```

- `~/.vim/after/ftplugin/yaml.vim`

```
set tabstop=2 "Indentation levels every two columns
set expandtab "Convert all tabs that are typed to spaces
set shiftwidth=2 "Indent/outdent by two columns
set shiftround "Indent/outdent to nearest tabstop
```

Project Layout

```
ansible
├── files
│   └── rsyslog-haproxy.conf
├── group_vars
│   ├── all/
│   ├── appcluster.yml
│   ├── app.yml
│   ├── bastion.yml
│   ├── cluster/
│   ├── db.yml
│   ├── loadbalancer/
│   ├── private_net.yml
│   ├── publichosts.yml
│   └── web.yml
├── inventory
│   ├── cloud-hosts
│   └── openstack.yml
├── lb-host.yml
├── app-blue-green-upgrade.yml
├── app-rolling-upgrade.yml
├── blue-green-start-switch.yml
├── deploy.yml
├── provision-hosts.yml
```

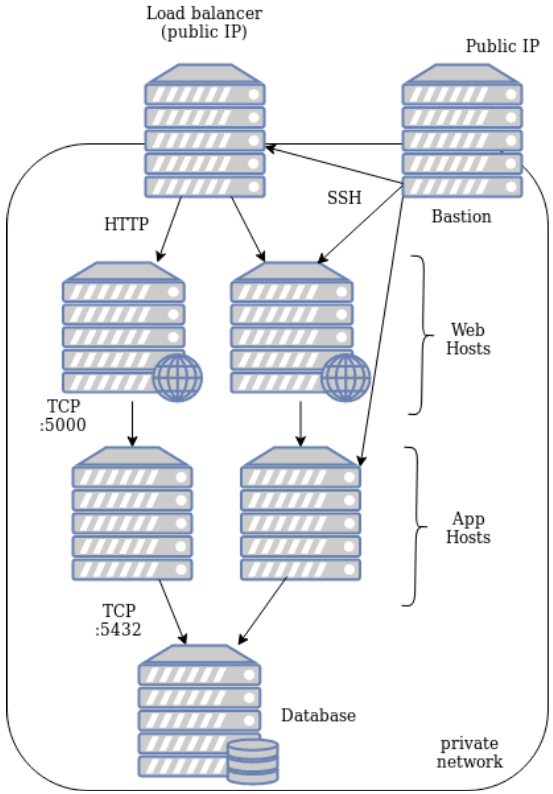
Provisioning Hosts

Creating Our Cluster

- For this tutorial we are going to need seven machines
  - 2 for nginx web server
  - 2 for our web application
  - 1 database host
  - 1 load balancer
  - 1 bastion
- Seems like a lot, but we are trying to simulate upgrades across a cluster

Our cluster

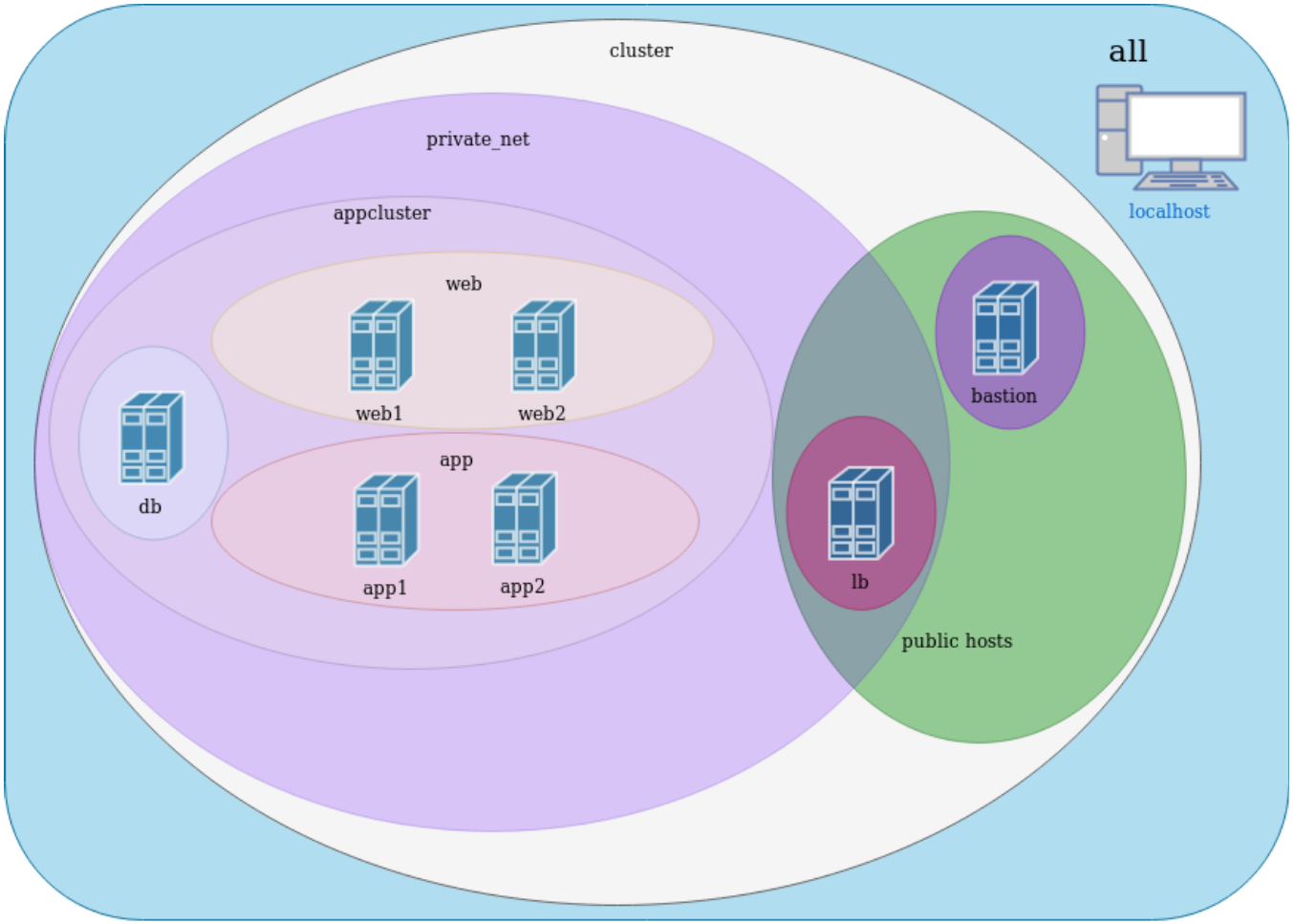
- HTTP traffic reaches web hosts via load balancer
- Application receives traffic from web hosts on port 5000
- DB receives traffic from app hosts on 5432
- SSH traffic
  - only bastion reachable from outside
  - all other hosts only SSH from bastion



Inventory Host Grouping

- Architecture defined using Ansible *groups*

```
ansible/inventory/cloud-hosts
```



Host Inventory

- Inventory and groups fundamental to deploying and configuring

```
hosts: web
```

```
localhost ansible_connection=local ansible_python_interpreter="/usr/bin/env

[appcluster]
pycon-web[1:2]
pycon-app[1:2]
pycon-db

[loadbalancer]
pycon-lb

[bastion]
pycon-bastion

[web]
pycon-web[1:2]

[db]
pycon-db

[app]
pycon-app[1:2]

[publichosts:children]
bastion
loadbalancer

[private_net:children]
appcluster
loadbalancer

[cluster:children]
bastion
loadbalancer
appcluster
```

```
ansible
├── group_vars/
│   ├── all/
│   ├── appcluster.yml
│   ├── app.yml
│   ├── bastion.yml
│   ├── cluster/
│   ├── db.yml
│   ├── loadbalancer/
│   ├── private_net.yml
│   ├── publichosts.yml
│   └── web.yml
```

The `provision-hosts.yml` playbook

`ansible/provision-hosts.yml`

- Tasks in the first play are executed on local machine

```
name:  Provision a set of hosts in Catalyst Cloud
hosts: localhost
gather_facts: false
```

- To run the playbook:

```
ansible-playbook -i ansible/inventory/cloud-hosts ansible/provision-hosts.yml
```

- We need to add a few things for this playbook to run

Using predefined variables

`groups`

- A dictionary representation of the inventory file

```
"groups": {
    "all": [
        "localhost",
        "pycon-web1",
        "pycon-web2",
        "pycon-db",
        "pycon-app1",
        "pycon-app2",
        "pycon-lb",
        "pycon-bastion"
    ],
    "app": [
        "pycon-app1",
        "pycon-app2"
    ],
    "appcluster": [
        "pycon-web1",
        "pycon-web2",
```

Define hosts

- Need to define hosts that we plan to create
- Hosts in the *cluster* group
- Add following to `provision-hosts.yml`

```
# ADD hosts to create
host_set: "{{ groups.cluster }}"
```

- Re-run the playbook

Cloud Modules

- Cloud deployments typically involve creating resources with a provider
  - Instances
  - Networks
  - Security groups, acls, etc
- We are using OpenStack Modules
  - The modules that start with os_

Setting up cloud resources

- Behind the scenes using the OpenStack API
  - same endpoints used by `openstacksdk` CLI
- Boilerplate for creating multiple cloud hosts
  - log in to cloud provider
  - create router, network, security groups
  - create each host

Using OpenStack cloud modules

- The first play uses cloud modules to create objects on your tenant

```
# ADD create cloud resource methods
- name: Connect to Catalyst Cloud
  os_auth:

- name: Create keypair
  os_keypair:
    name: "{{ keypair_name }}"
    public_key: "{{ ssh_public_key }}"

- name: Create network
  os_network:
    name: "{{ network_name }}"
    state: present

- name: Create subnet
  os_subnet:
    name: "{{ subnet_name }}"
    network_name: "{{ network_name }}"
    state: present
    cidr: "{{ subnet_cidr }}"
    allocation_pool_start: "{{ subnet_dhcp_start }}"
    allocation_pool_end: "{{ subnet_dhcp_end }}"
    ip_version: "4"
    dns_nameservers: "{{ default_nameservers }}"

- name: Create router
  os_router:
    state: present
    name: "{{ router_name }}"
    network: "{{ public_net_name }}"
    interfaces:
      - "{{ subnet_name }}"
```

Using predefined variables

`hostvars`

- Contains dictionary mapping all hosts to variables defined under `group_vars`

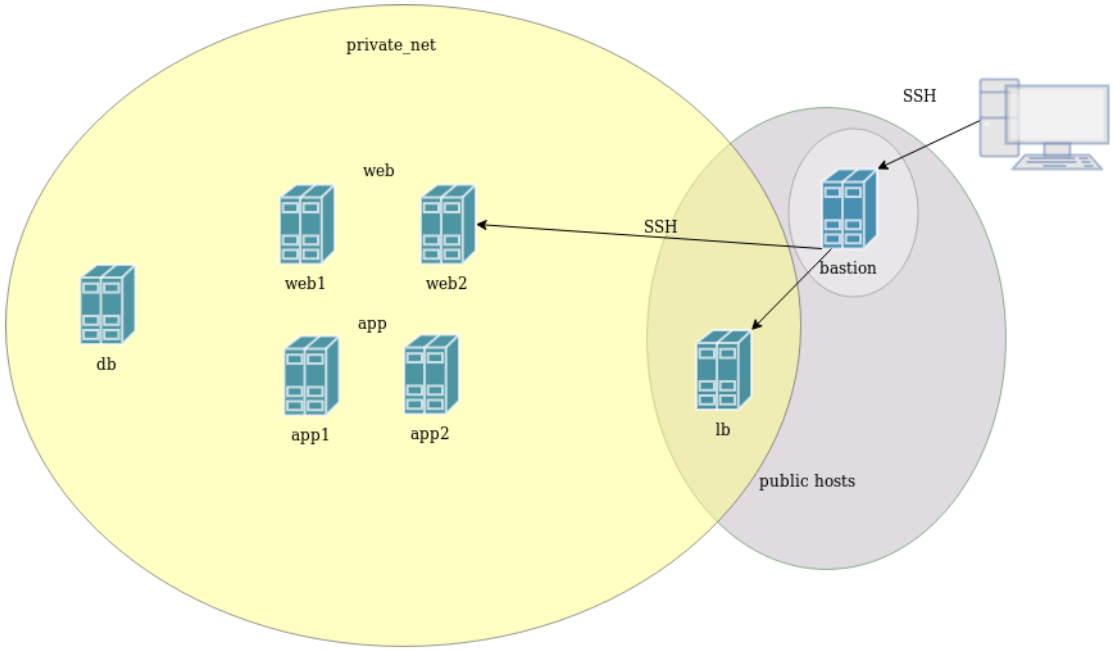
```
group_vars/
  app.yml
  db.yml
```

- Need to create list of security groups from all hosts
- Add following to playbook

```
# ADD extract security groups from inventory
security_groups: "{{ host_set | map('extract', hostvars, 'security_groups') | sum(start=[]) | list | unique }}"
security_group_names: "{{ security_groups | map(attribute='group') | list | unique }}"
```

Using Ansible via a bastion host

- Hosts in *private_net* group do not have public IP
- Only bastion is directly accessible by SSH
- All other hosts can only be reached from *bastion*



Traversing a bastion host

- Ansible relies on SSH to talk to remote hosts
- Just need to pass SSH arguments for hosts in *private_net* group
- Use `add_host` to assign `ansible_ssh_common_args`

```
# ADD SSH args
- name: Set ssh args for bastion
  add_host:
    name: "{{ item.openstack.name  }}"
    ansible_ssh_common_args: "-o StrictHostKeyChecking=no -o ForwardAgent=yes"
  loop: "{{ launch.results }}"
  when: item.openstack.name in groups.bastion

- name: Set ssh args for rest of cluster
  add_host:
    name: "{{ item.openstack.name  }}"
    ansible_ssh_common_args: >
      -o StrictHostKeyChecking=no
      -o ForwardAgent=yes
      -o ProxyCommand='ssh {{ hostvars[item.openstack.name].ansible_user }}@{{ hostvars[groups.bastion[0]].ansible_host }} exec nc -w300 %h %p'
  loop: "{{ launch.results }}"
  when: item.openstack.name in groups.private_net
```

Additional setup for hosts

- Edit `/etc/hosts` on each host
  - bastion host to resolve all hosts in cluster for SSH
  - *web* to resolve *app* host
  - *app* host to resolve *db*
- Set NZ locale, timezone, etc.

Resolving for SSH

- Add following to `provision-hosts.yml`

```
# ADD bastion -> private_net for SSH
- name: Set up the bastion host mapping
  hosts: bastion
  become: true
  tasks:
    - name: Add entry to /etc/hosts for all instances
      lineinfile:
        dest: /etc/hosts
        line: "{{ hostvars[item].ansible_host }} {{ item }}"
      with_items: "{{ groups.private_net }}"
```

Resolving application services

- Set up resolution for application components (i.e. nginx and application)

```
# ADD web -> app for proxy pass
- name: Set up web hosts with mapping to backend
  hosts: web
  become: true
  tasks:

    - name: Map each frontend host to speak to a specific backend
      lineinfile:
        dest: /etc/hosts
        line: "{{ hostvars[groups.app[(group_index | int) - 1]].ansible_host }} backend"

# ADD app -> db for application
- name: Add mapping for db on app boxes
  hosts: app
  become: true
  tasks:

    - name: Map each app host to speak to db
      lineinfile:
        dest: /etc/hosts
        line: "{{ hostvars[item].ansible_host }} {{ item }}"
      with_items: "{{ groups.db }}"
```

Set up locale and timezone

```
# ADD locale and timezone
- name: Set locale and local timezone
  hosts: cluster
  become: true
  tasks:

    - name: Add NZ locale to all instances
      locale_gen:
        name: en_NZ.UTF-8
        state: present

    - name: Set local timezone
      timezone:
        name: Pacific/Auckland
```

Provisioning Hosts

- Start the provisioning playbook

```
ansible-playbook -i ansible/inventory/cloud-hosts ansible/provision-hosts.yml
```

- Should take a few minutes to set up cluster
- In case task fails with SSH error just hit CTRL - C and restart

`ansible-inventory`

- Useful to gather info about hosts
  - public IP
  - groups
- Can use `ansible-inventory` to gather info about cluster

```
ansible-inventory --list
```

```
ansible-inventory --host pycon-bastion
```

- Pipe output through tools like `jq`

Get bastion IP

- We'll need the bastion IP if we want to SSH into hosts in the cluster

```
ansible-inventory --host pycon-bastion | jq '{"publicIP": .openstack.public_v4}'
```

```
{
  "publicIP": "202.49.242.143"
}
```

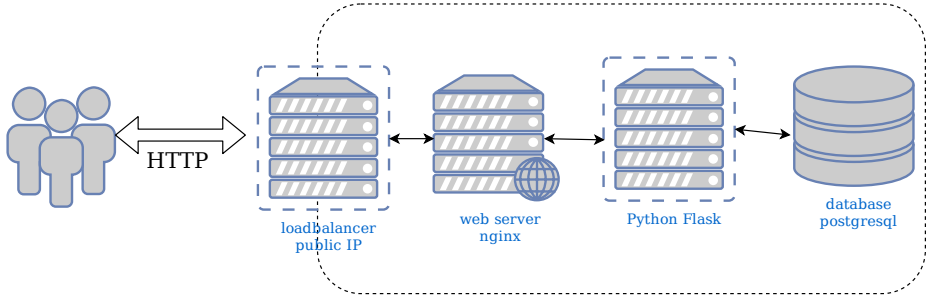
- Use this to SSH into cluster

```
ssh -A -t ubuntu@202.49.242.143 ssh pycon-web1
```

Deploying the application

Deploying our application

- The `deploy.yml` playbook sets up our application
  - Web server running nginx
  - App server running a Python Flask
  - Postgresql Database
  - HA proxy



Overview of deploy playbook

- `ansible-playbook --list-tasks <playbook` gives an overview of plays and tasks

```
ansible-playbook ansible/deploy.yml --list-tasks
```

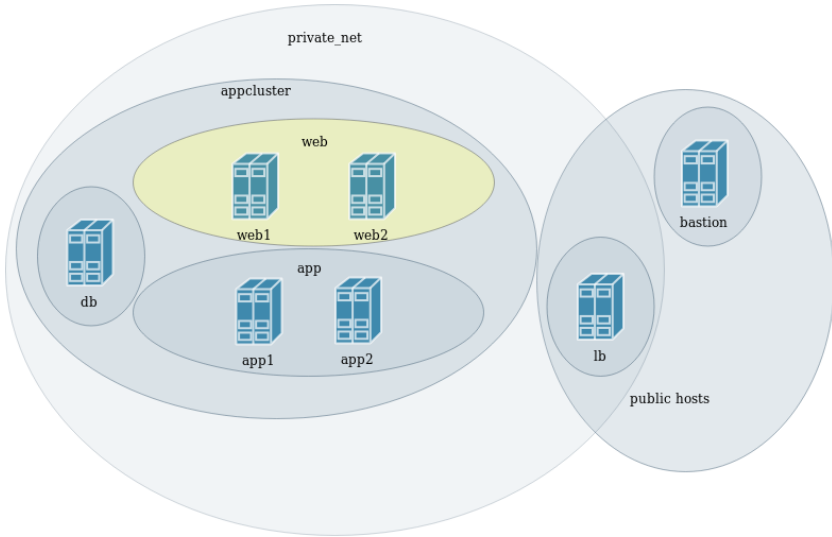
```
play #1 (private_net): Set ansible_host for private hosts     TAGS: []
  tasks:
  .
play #2 (cluster): Update apt cache on all machines   TAGS: []
  tasks:
  .

play #3 (db): Set up database machine TAGS: [deploy,db]
  tasks:
  .

play #4 (db): Set up app and database machine TAGS: [deploy,db]
  tasks:
  .

play #5 (app): Set up app server      TAGS: [deploy,app]
  tasks:
  .

play #6 (web): Set up nginx on web server     TAGS: [deploy,web]
  tasks:
  .
```

Role of Inventory and Groups

- *hosts* attribute influences which hosts Ansible interacts with

```
hosts: web
```

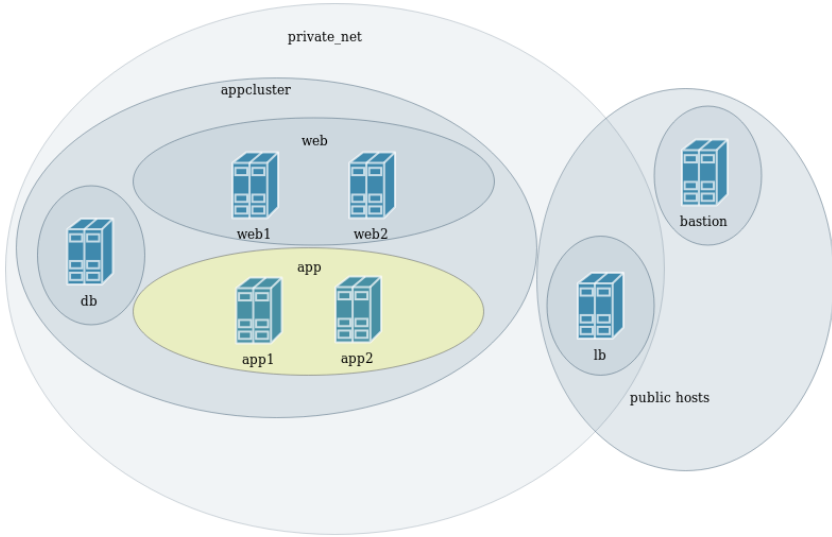
- This will interact with all hosts in the *web* group



Role of Inventory and Groups

```
hosts: app
```

- This will interact with all hosts in the *app* group



Deploying the application

- Run the deploy playbook

```
ansible-playbook ansible/deploy.yml
```

- Once deploy is finished you'll need the IP of your loadbalancer

```
ansible-inventory --host pycon-lb | jq '{"publicIP": .openstack.public_v4}'
```

- Should be able to open in your browser as:

```
http://<public ip>.xip.io/
```

Viewing HAProxy stats

- HAProxy provides an overview of active web hosts in cluster

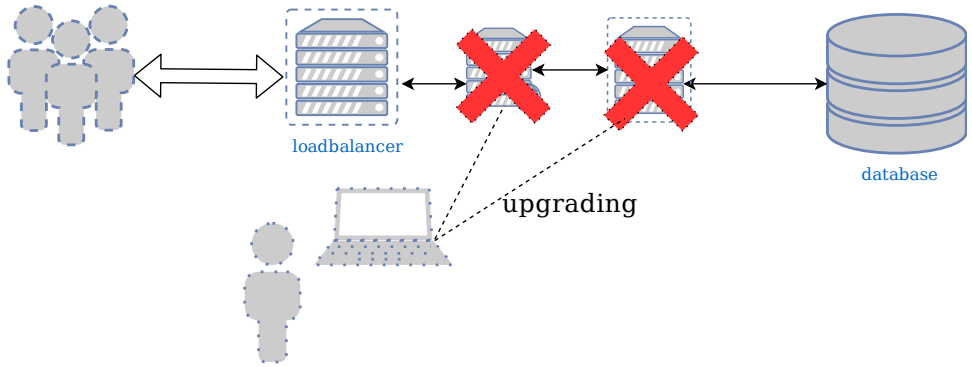
```
http://<public ip>.xip.io/haproxy?stats
```

- Login details
  - user: admin
  - password: train

Upgrade strategies

What can go wrong?

- Without some kind of redundancy, we risk of disrupting entire operation



Ideal upgrade scenario

- Minimal or zero downtime during upgrade of application
- Do not deploy a broken version of our application

Upgrade Strategies

- In-place rolling upgrade
- Blue-Green

In-place rolling upgrade

- Traditional approach to upgrading applications across a cluster
  - Creating new infrastructure can be prohibitively expensive
- Operates on infrastructure that already exists
- Minimise downtime by upgrading parts of the cluster at a time

Upgrading applications

```
ansible-playbook ansible/app-rolling-upgrade.yml -e app_version=v
```

- At the moment there is no real difference to running

```
ansible-playbook ansible/deploy.yml -e app_version=v2 --limit app
```

- Tempting to just rely on idempotent behaviour to *do the right thing*
- There are two problems with this approach
  - Ansible's default *batch management* behaviour
  - `deploy.yml` does not check *health* of application

Default *batch management* behaviour

- By default runs each task on all hosts concurrently
- A failed task might leave every host in cluster in a broken state

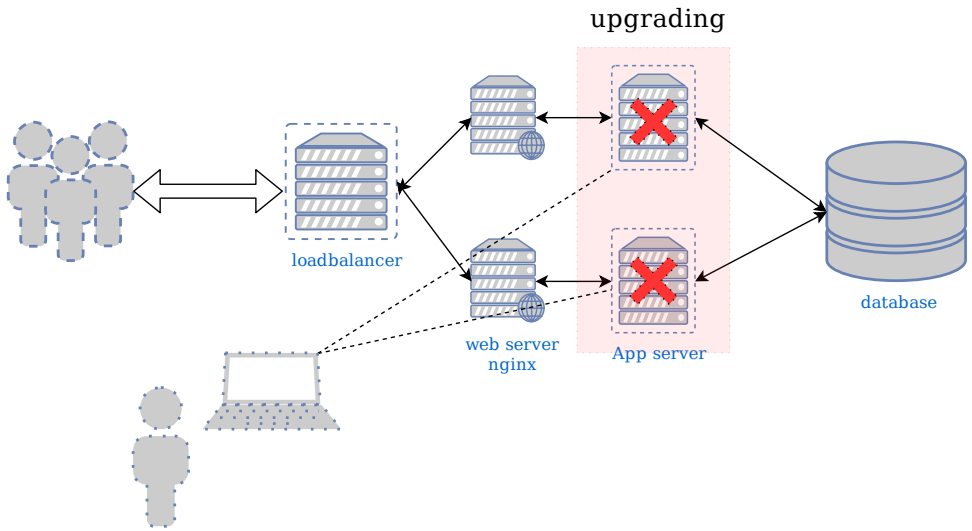
Tasks	Host1	Host2
task1	ok	ok
task2	ok	ok
task3	fail	fail
task4	-	-

```
TASK [Install app requirements] ******************************
Tuesday 20 August 2019  07:44:32 +1200 (0:00:02.565)   0:00:09
changed: [pycon-app1]
changed: [pycon-app2]

TASK [Install app requirements] ******************************
Tuesday 20 August 2019  07:44:32 +1200 (0:00:02.565)   0:00:09
failed: [pycon-app1]
failed: [pycon-app2]
```

Deploying broken code

- Broken code may not be obvious in task
- *One task leaves application in a broken state
- Run the rolling upgrade playbook with `app_version=v3`

Tasks	Host1	Host2
task1	ok	ok
task2	ok	ok
task3*	ok	ok
task4	ok	ok



Fixing the batch problem

- The `serial` attribute can be added to the *play* attributes
- Determines batch size Ansible will operate in parallel
  - integer
    - `serial: 1`
    - `serial: 3`
  - percentage of cluster
    - `serial: 50%`

Controlled batch size

- Running with `serial` attribute set to 1

Tasks	Host1	Host2
task1	ok	ok
task2	ok	ok
task3	ok	ok
task4	ok	ok

Using `serial` in our upgrade

- Run the `app-rolling-upgrade.yml` playbook again with `-e app_version=v1`
- Update `app-rolling-upgrade.yml` as follows:

```
- name: Upgrade application in place
  become: true
  hosts: app
  # Serial attribute
  serial: 1
```

- Try again with `-e app_version=v3`

Deploying broken code

- Deploying `app_version=v3` still breaks the application

Tasks	Host1	Host2
task1	ok	ok
task2	ok	ok
task3*	ok	ok
task4	ok	ok

Reset the environment

- Before we proceed, please reset your environment

```
ansible-playbook ansible/app-rolling-upgrade.yml -e app_version=v1
```

Failing fast

- Need to detect broken application and stop deployment
- Verify app is running after upgrade
- The Flask web application that runs on app server listens on port 5000
- Can use `wait_for` to stop and listen for port to be open before proceeding

Listen on port

- Add following to `app-rolling-upgrade.yml`

```
# ADD wait for 5000
- name: Make sure gunicorn is accepting connections
  wait_for:
    port: 5000
    timeout: 60
```

- We're still missing something so don't run the playbook yet!

Flushing handlers

- The application may not have loaded new configuration
- We need to force handler to restart gunicorn before waiting on port
- Add following to `app-rolling-upgrade.yml`

```
# ADD flush handlers
- meta: flush_handlers
```

- Now re-run the playbook with `-e app_version=v3`

Failing fast

- Playbook stops execution on first host when check on port fails

Tasks	Host1	Host2
task1	ok	-
task2	ok	-
restart gunicorn*	ok	-
wait_for	fail	-

Load balancing and upgrades

- During an upgrade we change configuration and restart the application
- Downtime might be disruptive to users of website
- Following update with `app_version=v3` half of the cluster is broken

```
curl --head http://<public ip>.xip.io
```

```
HTTP/1.1 502 Bad Gateway
```

Reset the environment

- Before we proceed, please reset your environment

```
ansible-playbook ansible/app-rolling-upgrade.yml -e app_version=v1
```

Avoiding disruptions

- Ideally the loadbalancer should not send traffic to the hosts(s) we are updating
- While upgrading *app* host, need to disable traffic to upstream web host

Host Context

- The `hosts:` attribute of a play determines *context*

```
- name: Play on app host
  hosts: app
```

- While on host `app1`, we can call all inventory variables by name, i.e.
  - `ansible_host`
- If we want variable for a different host, must use *hostvars*
  - hostvars['otherhost'].ansible_host

Delegation

- Sometimes need to configure one host *in the context of another host*
- Run a command on server **B** using inventory from **A**
  - enable/disable web hosts at the load balancer
- The `delegate_to` directive is useful for this

Using delegation

- We want to disable host we're updating on pycon-lb

```
# ADD disable application at lb
- name: Disable application at load balancer
  haproxy:
    backend: catapp-backend
    host: "{{ web_server }}"
    state: disabled
  delegate_to: "{{ item }}"
  loop: "{{ groups.loadbalancer }}"
```

Enabling host at loadbalancer

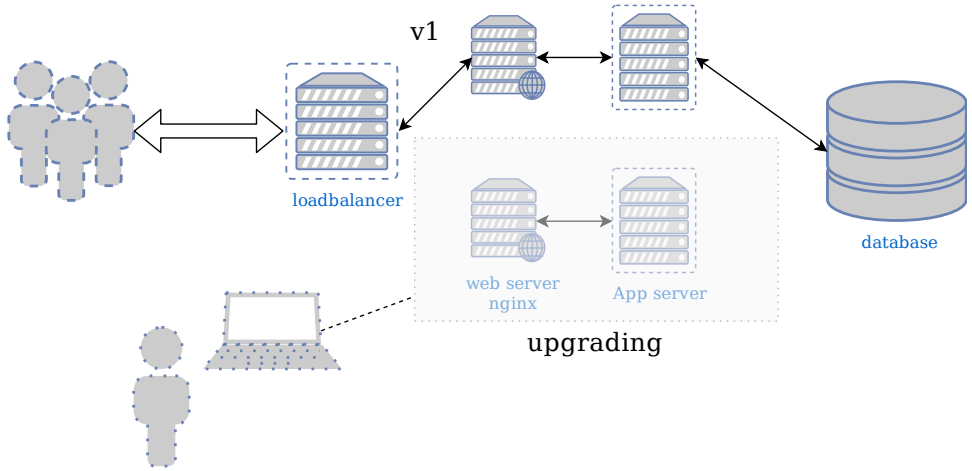
- When we are sure the app is running, we need to re-enable traffic to the host

```
# ADD enable application at lb
- name: Re-enable application at load balancer
  haproxy:
    backend: catapp-backend
    host: "{{ web_server }}"
    state: enabled
  delegate_to: "{{ item }}"
  loop: "{{ groups.loadbalancer }}"
```

In Place Rolling Upgrade

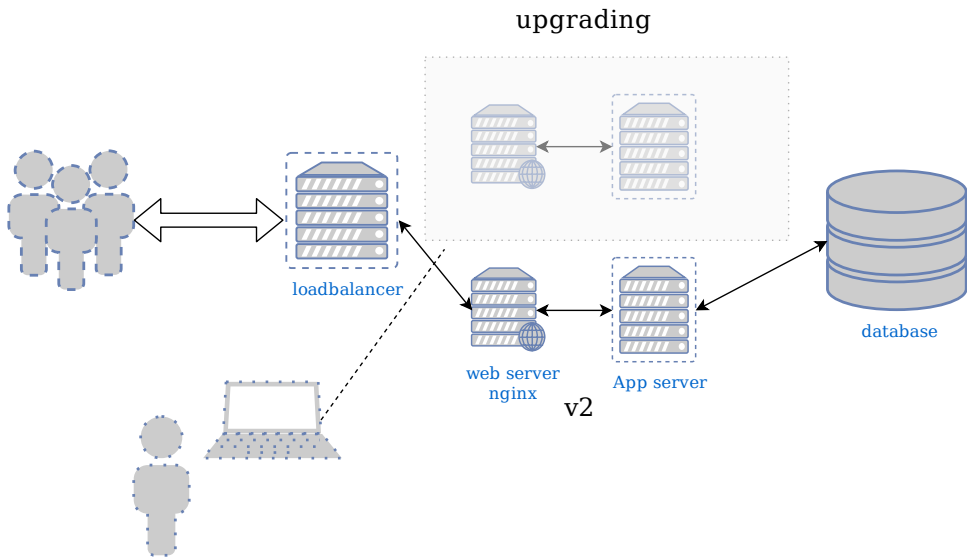
- Run playbook with `app_version=`
  - v1
  - v2
  - v3
- During upgrades
  - curl site url from terminal
  - check HAProxy stats
- Upgrade to v3 should not leave entire cluster (or part of site) broken

First step of in place upgrade



- Disable application at LB (no HTTP requests)
- Upgrade necessary applications, configuration
- Re-enable at LB

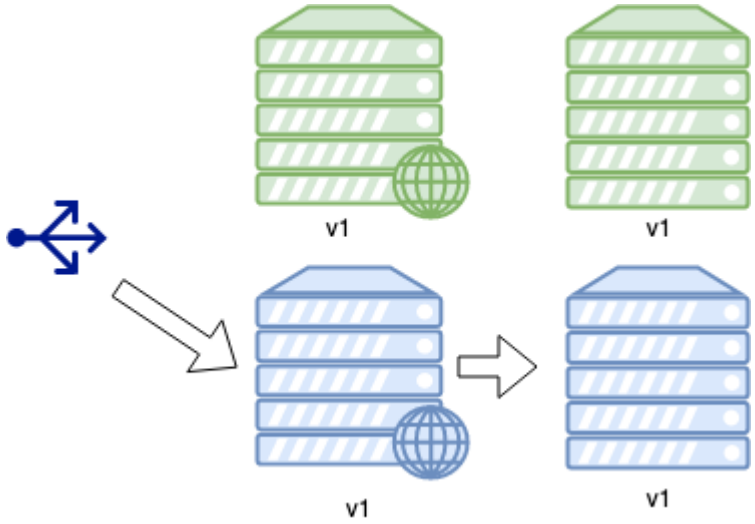
In place rolling upgrade



- Repeat process across pool
- Mixed versions will be running for a period of time

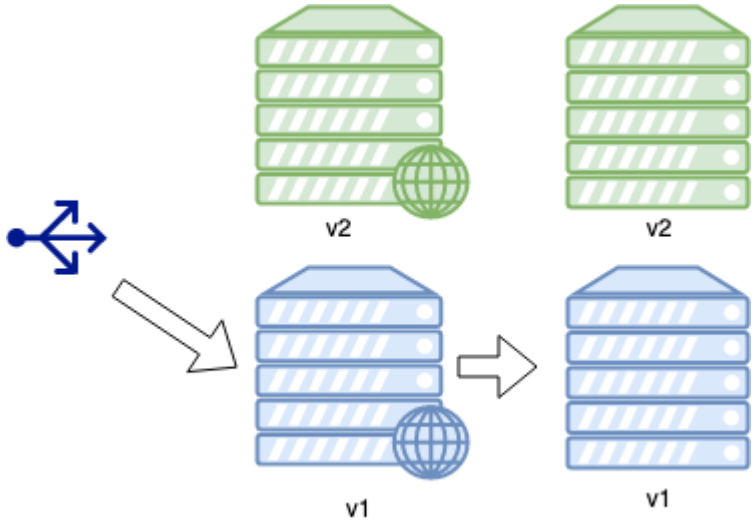
Blue Green Deployments

Blue Green Deployments



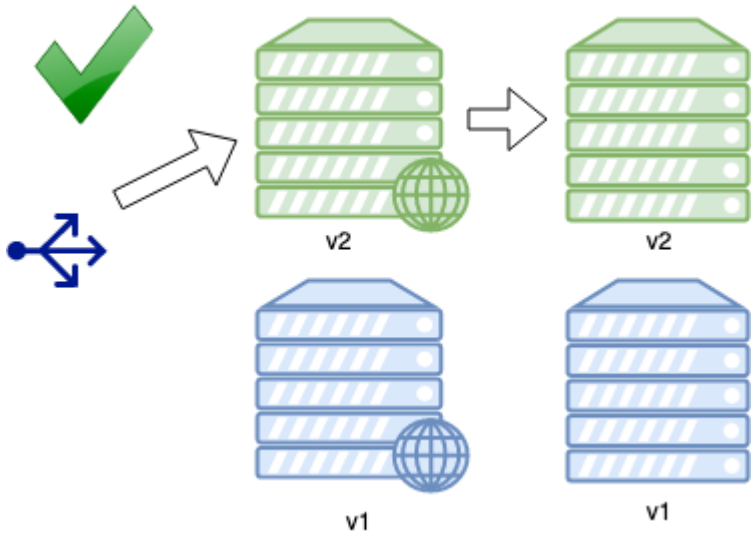
Start with all traffic going to *blue* hosts

Begin update



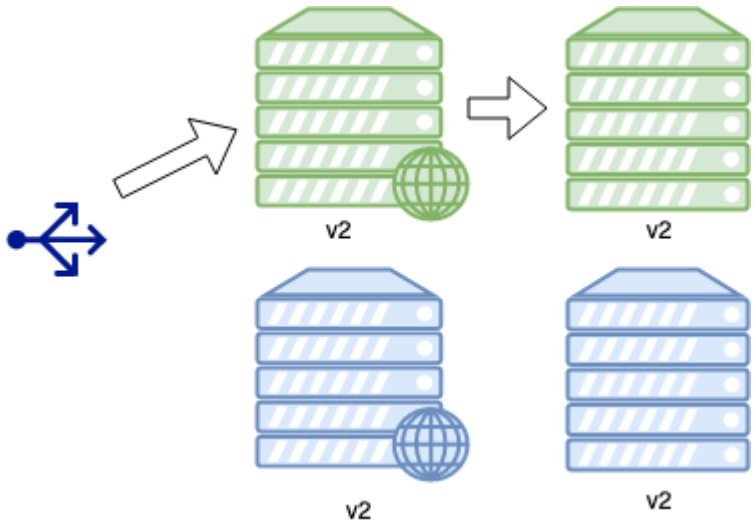
Update traffic on *green* hosts

Check green is ok



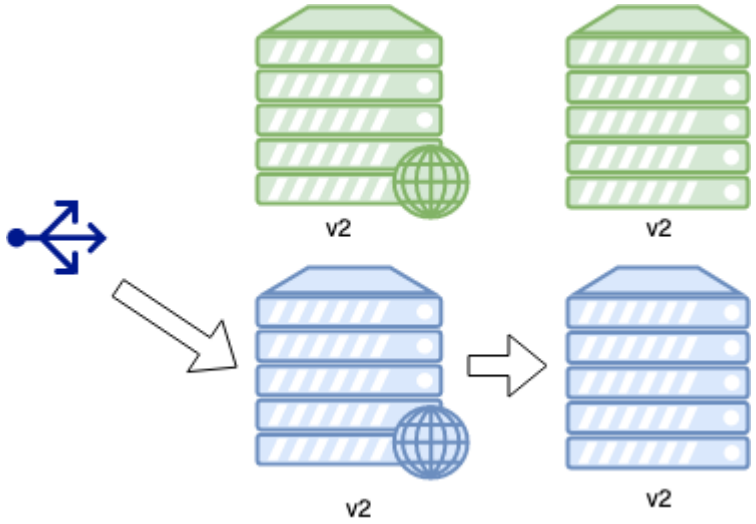
Direct traffic through green and verify ok

Update blue side of cluster



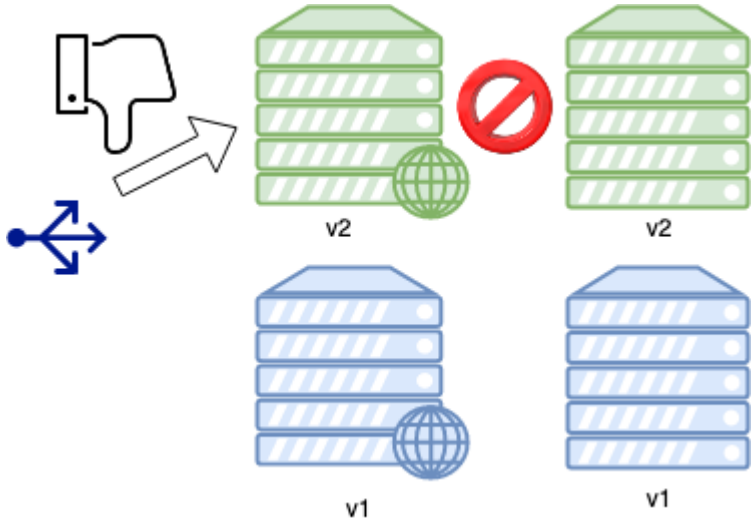
Ok, so update *blue* side

Reset traffic to blue



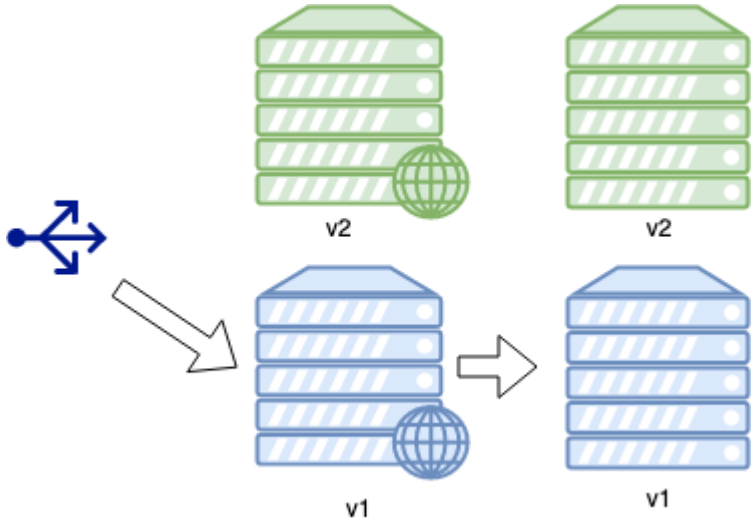
Reset traffic to *blue* side of cluster

Dealing with failure



Alternative if green app not healthy

Return to original state



Redirect traffic back to *blue*

Setting up Blue-Green

- We need to do is put our cluster in *blue-green* mode
- First reset our environment

```
ansible-playbook ansible/app-rolling-upgrade.yml -e app_version=v1
```

- Run the following playbook:

```
ansible-playbook ansible/setup-blue-green.yml -e live=blue
```

- Verify half of cluster active on HAProxy stats page

Blue green update playbook

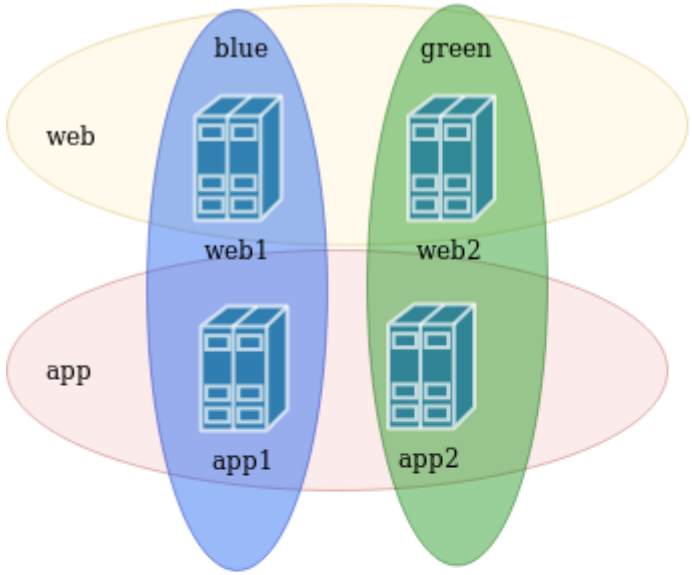
- For blue green we will use the following playbook

```
ansible/app-blue-green-upgrade.yml
```

- In our inventory

```
# ansible/inventory/cloud-hosts
[blue]
pycon-web1
pycon-app1

[green]
pycon-web2
pycon-app2

[blue_green:children]
blue
green
```

Inventory and groups



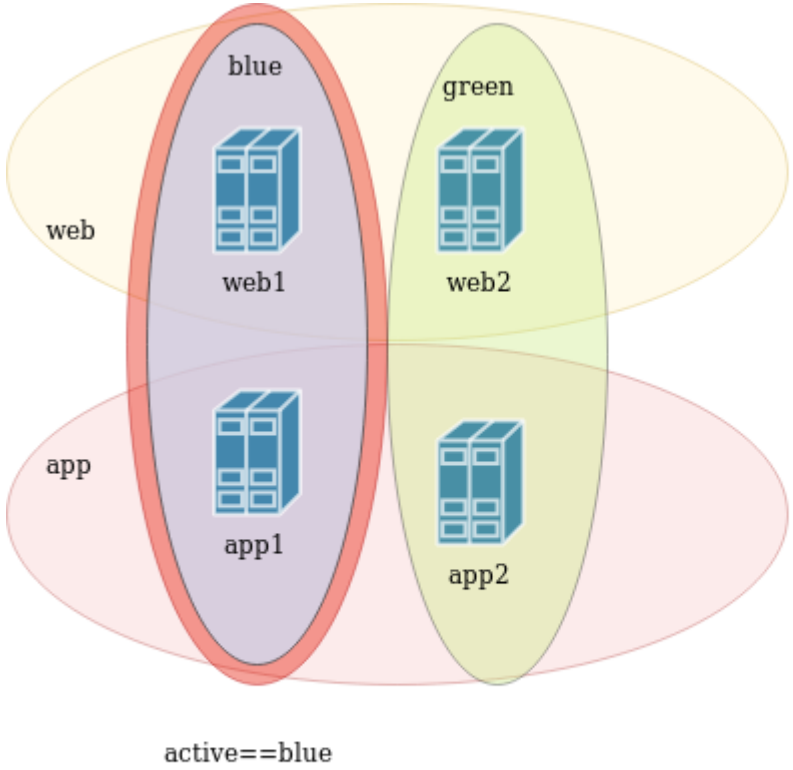
Ad hoc groups

- For *blue-green* we need to assign *active* and *not active* half of cluster
- Can assign groups of hosts to *ad hoc* groups
- By default we declare *blue* active
- Add following to `app-blue-green-upgrade.yml`

```
 # ADD set active group
- name: Set live group as active
  hosts: localhost
  gather_facts: false
  vars:
    active: "{{ groups[ live | default('blue') ] }}"
  tasks:
    - name: Add active hosts to group
      add_host:
        name: "{{ item }}"
        groups:
          - active
      with_items: "{{ active | default(groups.blue_green) }}"
```

Operating on groups and subgroups

- The play we added creates an *ad hoc* group called **active**
- Initially equal to **blue** group
- We want to update hosts **not in the active**



Group set theory

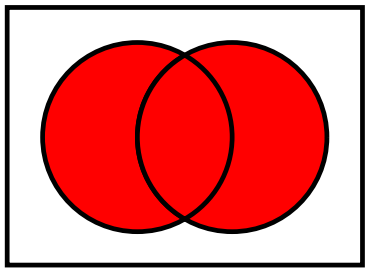
Ansible set theory operators

- The *hosts* attribute has syntax for set theory operations on inventory
- These enable fine control over which hosts playbooks operate

Union

A ∪ B

Combination of hosts in two groups



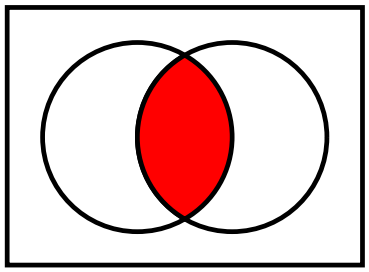
All hosts in *web* and *db* groups

```
- name: Union of hosts
  hosts: web:db
  tasks:
```

Intersection

A ∩ B

Hosts that are in first and second group



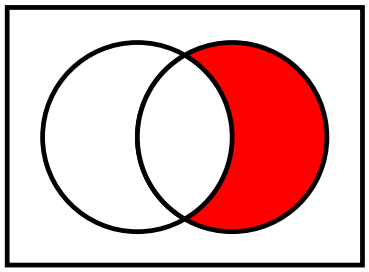
Hosts that are in both the *web* and the *blue* group

```
- name: Intersection of hosts
  hosts: web:&blue
  tasks:
```

Difference

A \ B

Set of hosts in first set but not in second set



Hosts that are in the *app* group **but not** in the *active* group

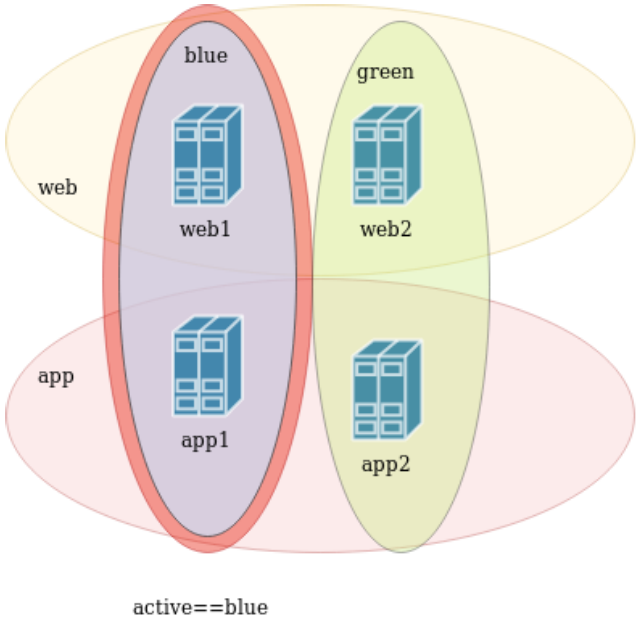
```
- name: Difference of groups
  hosts: app:!active
  tasks:
```

Set operators and upgrade

- Update playbook similar to rolling upgrade example
- Update app in inactive part of cluster

```
# ADD set to update
hosts: app:!active
```

- Should update app2



Verify app is running

- Restart the app and verify it is listening on port

```
# ADD flush handlers and check port
- meta: flush_handlers

- name: Make sure gunicorn is accepting connections
  wait_for:
    port: 5000
    timeout: 60
```

Enabling traffic to green

- Use delegation to enable traffic to green at loadbalancer

```
# ADD enable traffic to inactive
- name: Enable traffic to updated app server
  hosts: web:!active
  become: true
  tasks:
    - name: Enable application at load balancer
      haproxy:
        backend: catapp-backend
        host: "{{ inventory_hostname }}"
        state: enabled
      delegate_to: "{{ item }}"
      loop: "{{ groups.loadbalancer }}"
```

Stop traffic to blue

- Now disable blue side at loadbalancer

```
# ADD disable traffic to active side
- name: Stop traffic to initial live group
  hosts: web:&active
  become: true
  tasks:
    - name: Disable application at load balancer
      haproxy:
        backend: catapp-backend
        host: "{{ inventory_hostname }}"
        state: disabled
      delegate_to: "{{ item }}"
      loop: "{{ groups.loadbalancer }}"
```

Run blue green upgrade

- Let's run the blue green upgrade playbook

```
ansible-playbook ansible/app-blue-green-upgrade.yml -e app_version=v2
```

- Can switch back to blue active by running

```
ansible-playbook ansible/setup-blue-green.yml -e live=blue
```

- Try running upgrade with v3 and v4

Additional check

- May want to make additional checks on site
- v4 works but does not display version on site
- Add additional check to play

```
# ADD check version display
- name: Check that the site is reachable via nginx
  uri:
    url: "http://{{ ansible_host }}:5000"
    status_code: 200
    return_content: yes
    headers:
      HOST: my-app.cats
  register: app_site
  failed_when: "'version: ' + app_version not in app_site.co
  delegate_to: "{{ web_server }}"
```

The End

- Please do not forget to clean up your clusters!

```
ansible-playbook ansible/remove-hosts.yml
```