

Lecture 19: Deployment/Ansible

AC215

Pavlos Protopapas

SEAS/ Harvard



Announcements

- **Showcase Info - Missing ~7 projects**

Form  <https://forms.gle/CewUpMnmYq2BxupW6>

- **React Zoom Session -
Friday 11/15 - 3:00 - 4:30 PM (will be recorded)**

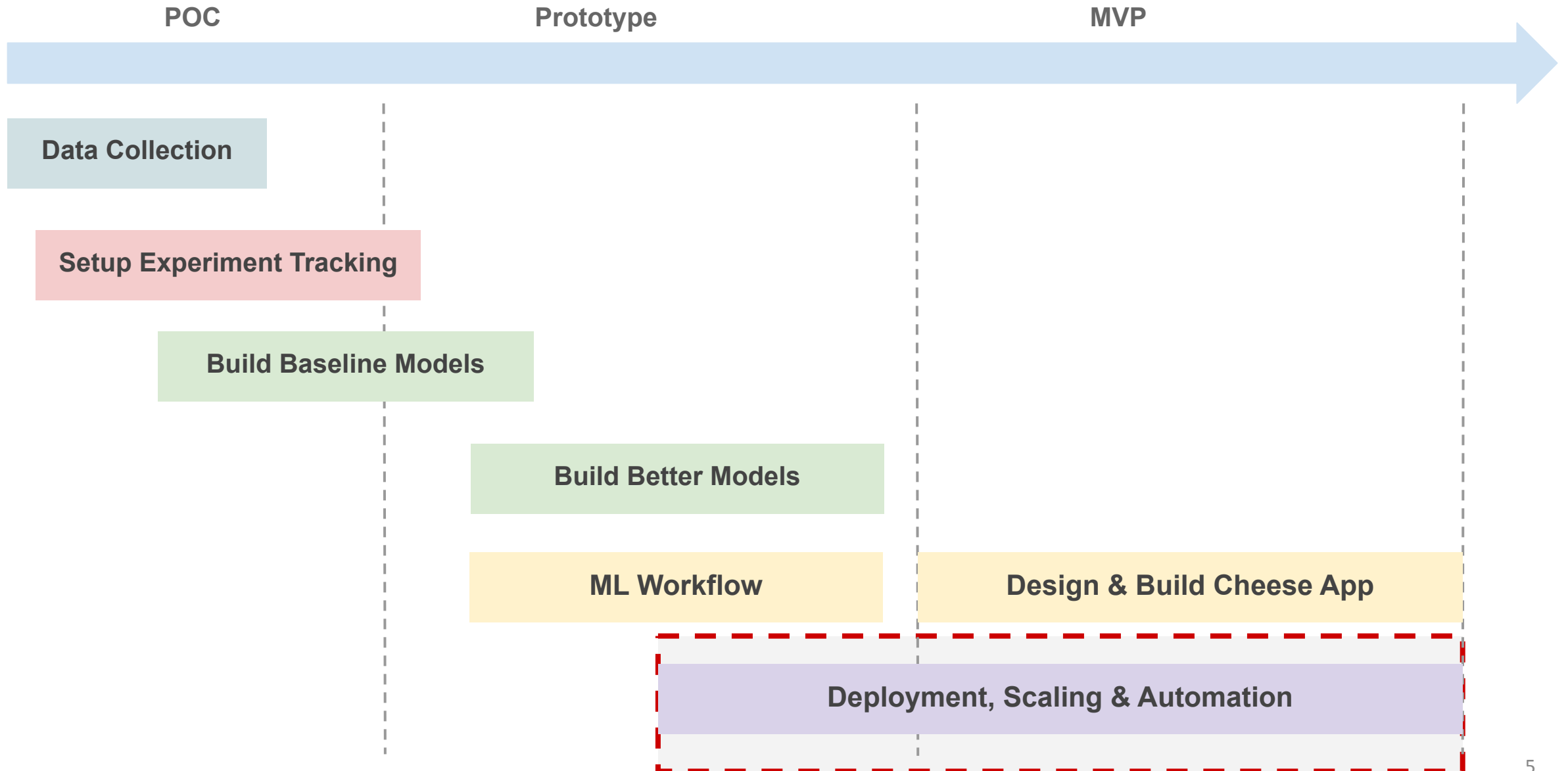
Outline

1. Recap
2. Deployment

Outline

1. **Recap**
2. Deployment

Recap: Cheese App Status



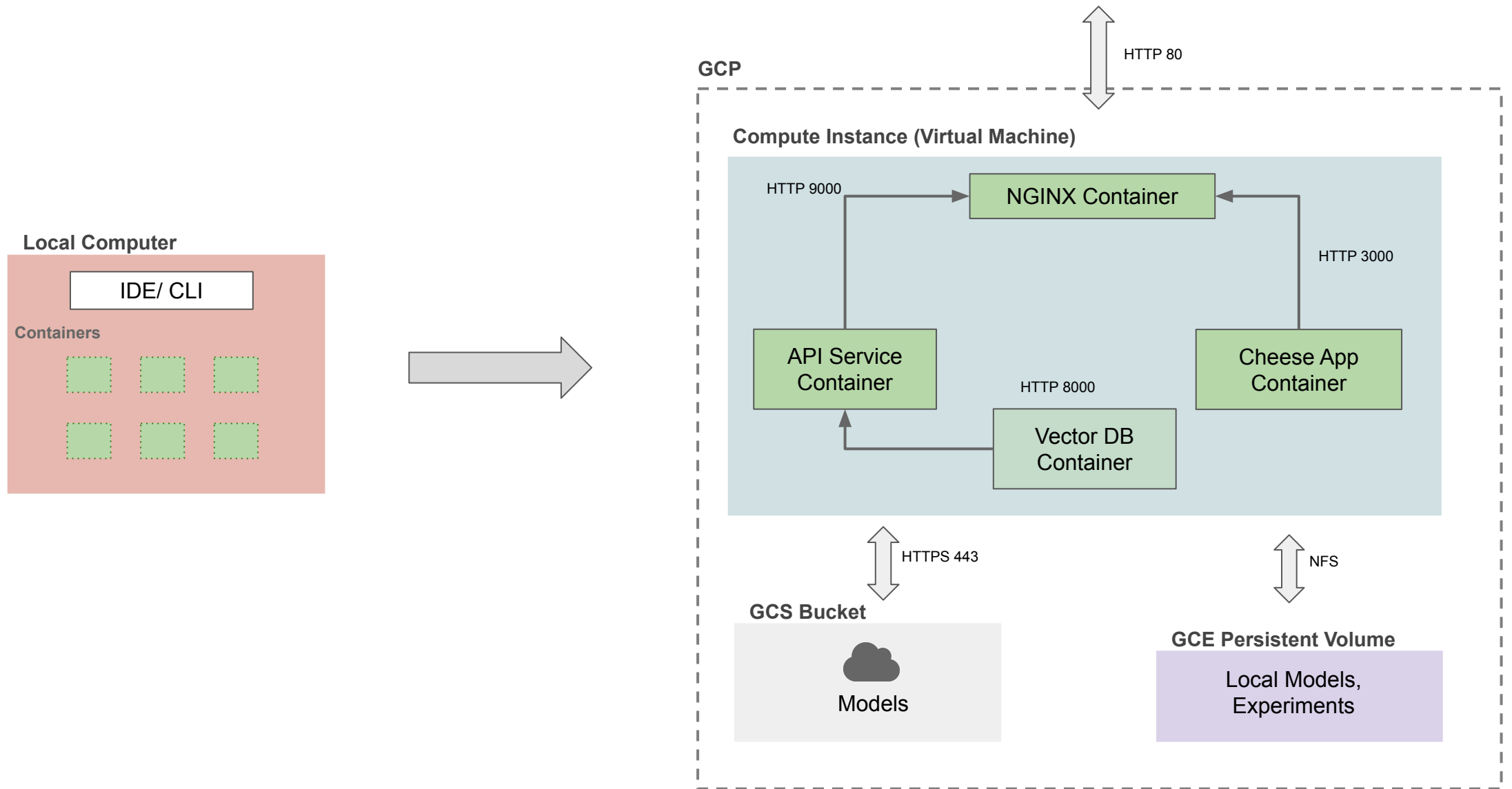
Recap: APIs & Frontend App

- Everything we built is on our **local computer**
- We need to **deploy** this to a server so our users can access

Outline

1. Recap
- 2. Deployment**

Deployment: Goal



Tutorial: Deployment Steps (Manual)

Push to Docker Hub: Build and push Docker images for API service and Frontend to Docker Hub repository

Setup VM Environment: Create VM instance on GCP and install Docker, create required folders and set permissions

Deploy Containers: Run Docker containers for API service and Frontend, creating a dedicated network for communication

Configure Web Server: Setup Nginx as reverse proxy, configure routing for API and Frontend services

[Cheese App - Deployment to GCP \(Manual\)](#)



Deployment Automation

In our manual deployment there were various steps to keep track of.

We want to **automate** this!



A N S I B L E

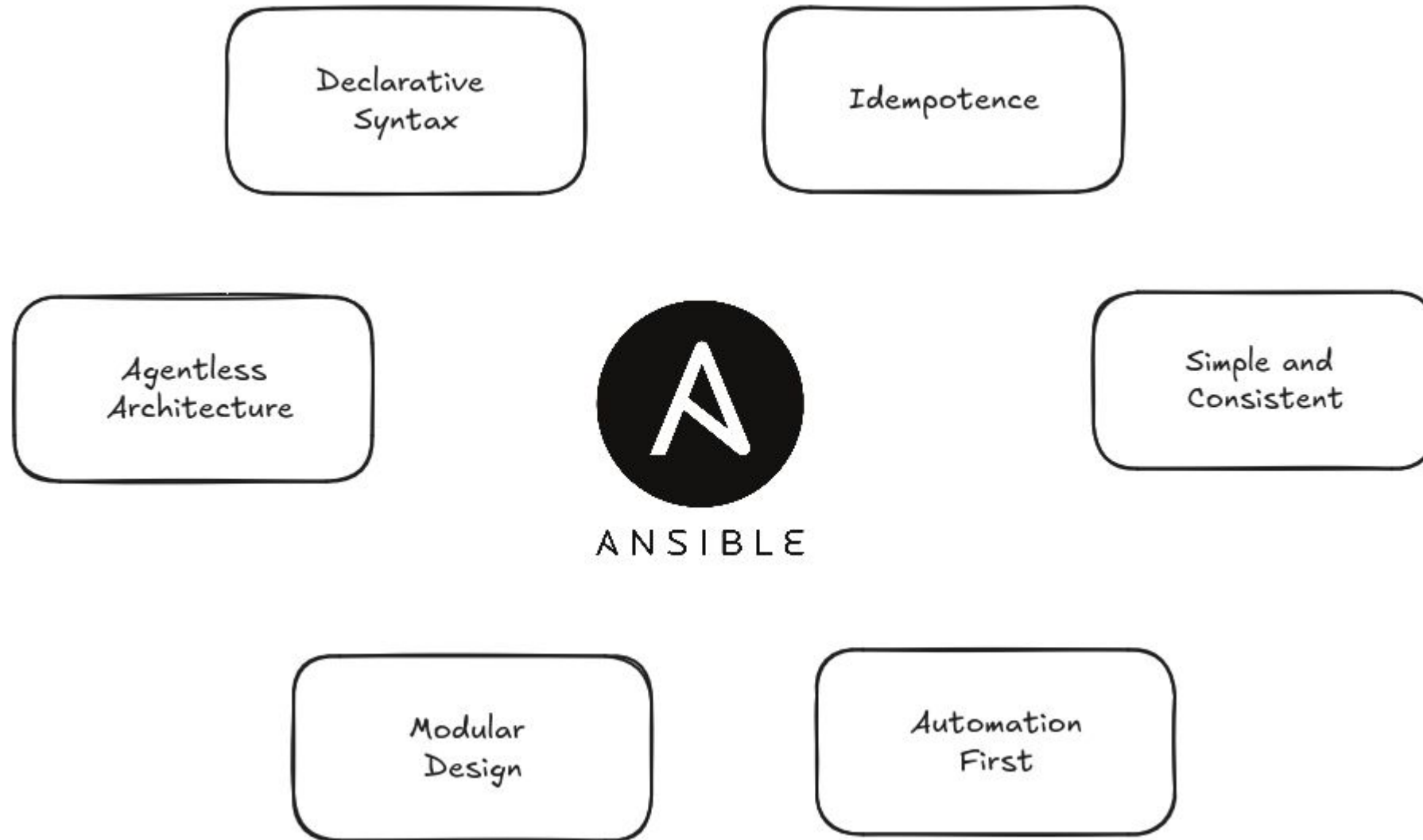
Ansible

- Is a tool for infrastructure automation
- Think of infrastructure as code
- Ansible scripts (playbooks) consist of instructions for tasks like
 - Server & Cluster creation/deletion
 - Software installation & setup
 - Networking setup
- Everything is code, so you can check it into GitHub and share

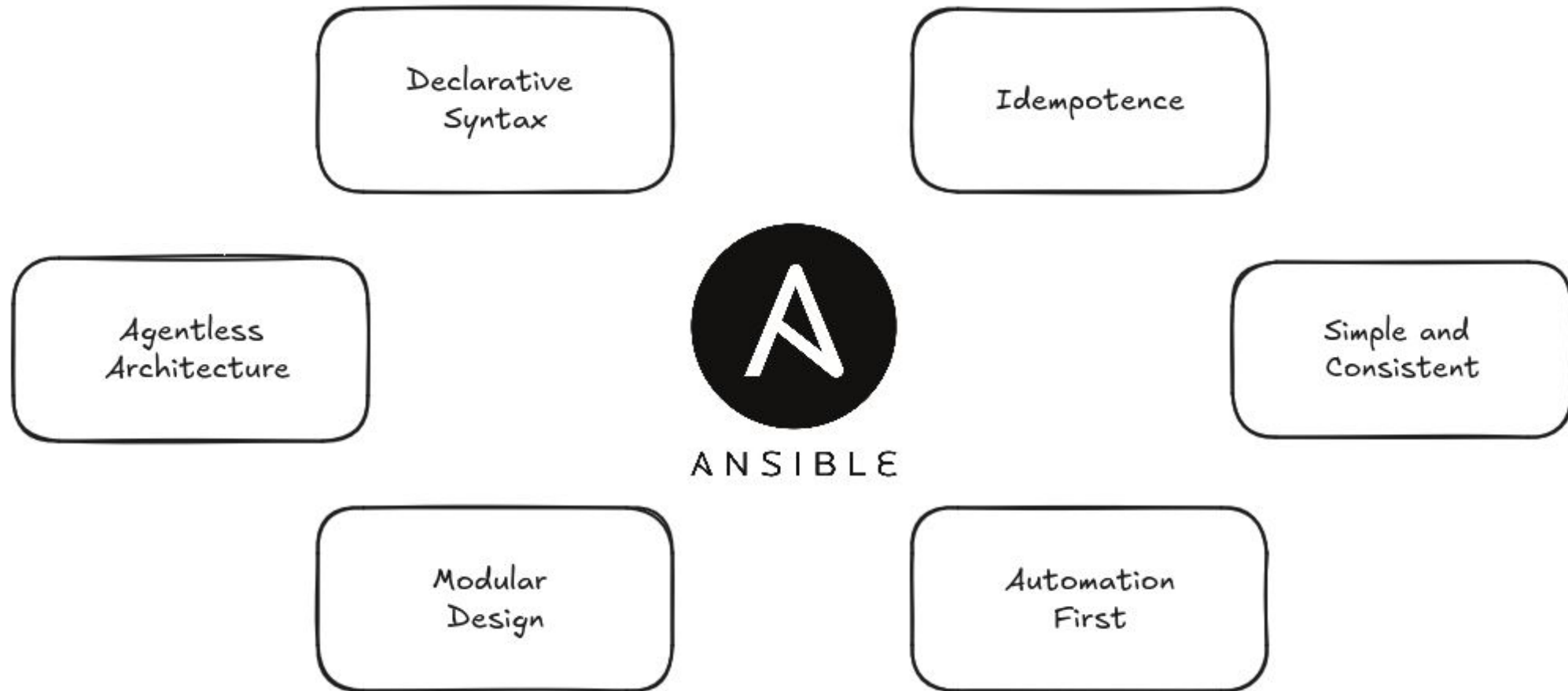
Ansible: History

- Ansible was created by Michael DeHaan in 2012 and acquired by Red Hat in 2015.
- It configures systems like Linux and Windows **without needing an agent**, using SSH or Windows Remote Management.
- The control node runs on any system with **Python**.
- System setup is defined in a simple **domain specific language** written in files called playbooks.

Ansible : Principles



Ansible : Principles



Ansible: Concepts

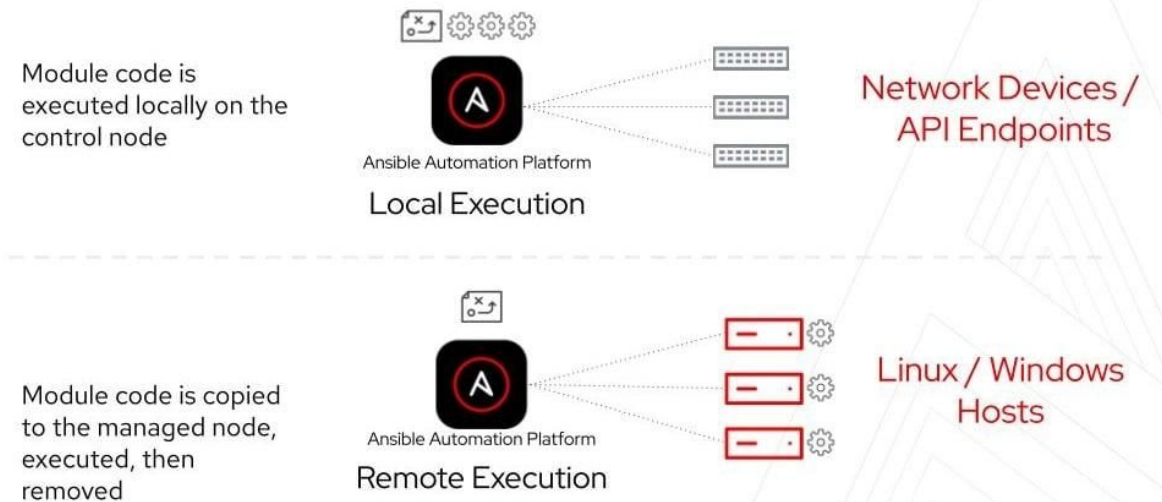
- Ansible commands
- Playbooks
- Inventory
- Fact Gathering
- Plays
- Tasks
- Modules

Ansible: Concepts

Ansible connects to managed nodes and pushes out small programs — called **modules** — to them.

Ansible **executes** these modules **remotely** (over SSH by default) and removes them when finished.

Network Automation compared to servers



ANSIBLE MANAGED HOSTS DON'T NEED ANY
INSTALLED AGENTS

Ansible : Commands

Running Ad-Hoc Commands with Ansible

```
ansible <target> -m <module> -a "<arguments>" -i <inventory>
```

- <target>: Specifies the host(s) or group(s) to run the command on.
- -m <module>: Defines the module to use (e.g., ping, shell, yum).
- -a "<arguments>": Provides arguments for the module (e.g., commands for shell).
- -i <inventory>: Points to the inventory file listing the hosts.

Ansible : Commands

Some useful adhoc commands

ping all your managed hosts

```
ansible all -m ping
```

list all managed hosts

```
ansible all --list-hosts
```

gather facts on managed hosts

```
ansible webservers -m gather_facts
```

Ansible : Commands

More useful adhoc commands

gather facts on specific host

```
ansible webservers -m gather_facts --limit 172.16.250.132
```

install apache2 on webservers with privilege escalation

```
ansible webservers -m apt -a name=apache2 --become --ask-become-pass
```

update apt cache on all servers with privilege escalation

```
ansible all -m apt -a update_cache=true --become --ask-become-pass
```

Ansible Playbook command

Running Playbooks with Ansible

```
ansible-playbook <playbook_file.yml> -i <inventory> [options]
```

- <playbook_file.yml>: YAML file with tasks and configurations to automate.
- -i <inventory>: Inventory file listing hosts or groups.
- [options]: Additional options, such as --check (dry-run) or --limit (run on specific hosts).

Ansible : Commands

Adhoc commands become tasks in plays/playbook

We run plays/playbooks via ansible command

```
ansible-playbook --ask-become-pass site.yml
```

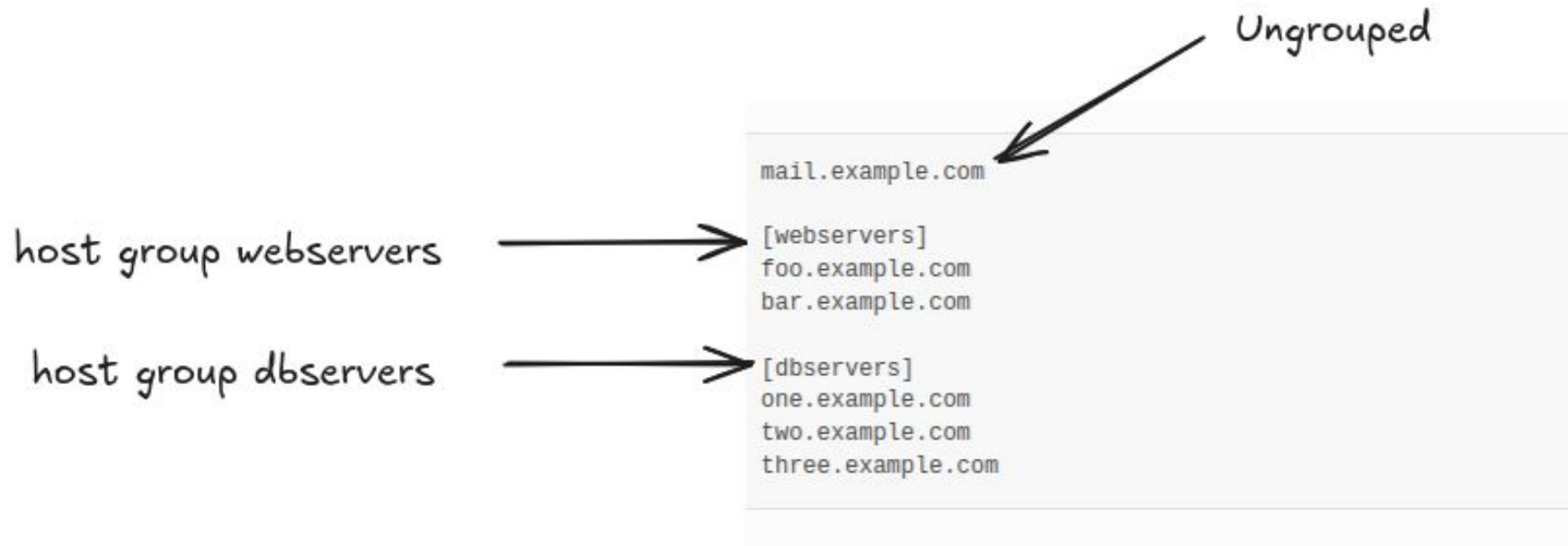
Ansible : Playbooks

- Ansible Playbooks manage **system behavior** written in a Domain Specific Language (DSL) based on **YAML** (Yet Another Markup Language)
- The **declarative** syntax is easy to read and supports modular design.
- Playbooks are highly **sensitive** to **indentation**.

```
---  
- name: Update web servers  
  hosts: webservers  
  remote_user: root  
  
  tasks:  
    - name: Ensure apache is at the latest version  
      ansible.builtin.yum:  
        name: httpd  
        state: latest  
  
    - name: Write the apache config file  
      ansible.builtin.template:  
        src: /srv/httpd.j2  
        dest: /etc/httpd.conf  
  
- name: Update db servers  
  hosts: databases  
  remote_user: root  
  
  tasks:  
    - name: Ensure postgresql is at the latest version  
      ansible.builtin.yum:  
        name: postgresql  
        state: latest  
  
    - name: Ensure that postgresql is started  
      ansible.builtin.service:  
        name: postgresql  
        state: started
```

Ansible : Inventory

- The inventory file lists **hosts** and organizes them into groups.
- Hosts are identified by domain names or IP addresses.
- **Groups** are defined by headers in the file.
- Hosts outside any group are placed in the “**ungrouped**” group.
- Ansible also includes a default group named “all,” containing all hosts.



Ansible : Modules

- **Modules** represent the desired state of the system.
- These modules are designed to be **idempotent** when possible, only making changes to a system when necessary.

You can find Ansible modules in the [Ansible Documentation](#), which lists all available modules by category. Additionally, modules are included in the standard Ansible installation and can be browsed directly on your system, typically located in `/usr/share/ansible/plugins/modules`.

The screenshot displays two pages from the Ansible documentation website. The top page, titled 'Indexes of all modules and plugins', provides a comprehensive list of plugin indexes categorized by type, such as Become Plugins, Cache Plugins, Callback Plugins, and more. The bottom page, titled 'ansible.builtin.package module – Generic OS package manager', features a 'Note' section explaining that this module is part of the 'ansible-core' collection and can be used with the short name 'package' or the fully qualified name 'ansible.builtin.package'. It also includes a table of contents with links to Synopsis, Requirements, Parameters, Attributes, Notes, and Examples, followed by a 'Synopsis' section describing the module's function in managing packages across different operating systems.

✖ / Indexes of all modules and plugins

This is the **latest** (stable) Ansible community documentation. For Red Hat Ansible Automation Platform subscriptions, see [Life Cycle](#) for version details.

Indexes of all modules and plugins

Plugin indexes

- [Index of all Become Plugins](#)
- [Index of all Cache Plugins](#)
- [Index of all Callback Plugins](#)
- [Index of all Cliconf Plugins](#)
- [Index of all Connection Plugins](#)
- [Index of all Filter Plugins](#)
- [Index of all Httpapi Plugins](#)
- [Index of all Inventory Plugins](#)
- [Index of all Lookup Plugins](#)
- [Index of all Modules](#)
- [Index of all Netconf Plugins](#)
- [Index of all Roles](#)
- [Index of all Shell Plugins](#)
- [Index of all Strategy Plugins](#)
- [Index of all Test Plugins](#)
- [Index of all Vars Plugins](#)

⏪ Previous

Next ⏩

✖ / Collection Index / Collections in the Ansible Namespace / [Ansible.Builtin](#) / [ansible.builtin.package](#) module – Generic OS package manager

[Edit on GitHub](#)

This is the **latest** (stable) Ansible community documentation. For Red Hat Ansible Automation Platform subscriptions, see [Life Cycle](#) for version details.

ansible.builtin.package module – Generic OS package manager

Note

This module is part of `ansible-core` and included in all Ansible installations. In most cases, you can use the short module name `package` even without specifying the `collections` keyword. However, we recommend you use the Fully Qualified Collection Name (FQCN) `ansible.builtin.package` for easy linking to the module documentation and to avoid conflicting with other collections that may have the same module name.

- [Synopsis](#)
- [Requirements](#)
- [Parameters](#)
- [Attributes](#)
- [Notes](#)
- [Examples](#)

Synopsis

- This module manages packages on a target without specifying a package manager module (like `ansible.builtin.dnf`, `ansible.builtin.apk`, ...). It is convenient to use in an heterogeneous environment of machines without having to create a specific task for each package manager. `ansible.builtin.package` calls behind the module for the package manager used by the operating system discovered by the module `ansible.builtin.setup`. If `ansible.builtin.setup` was not yet run, `ansible.builtin.package` will run it.
- This module acts as a proxy to the underlying package manager module. While all arguments will be passed to the underlying module, not all modules support the same arguments. This documentation only covers the minimum intersection of module arguments that all packaging

Ansible : Fact Gathering

gather facts on specific host

ansible webserver -m gather_facts --
limit 172.16.250.132

info can be used to define
variables or conditions later on

```
    "scheduler_mode": "deadline",
    "sectors": "167772160",
    "sectorsize": "512",
    "size": "80.00 GB",
    "support_discard": "0",
    "vendor": null,
    "virtual": 1
  }
},
"ansible_distribution": "CentOS",
"ansible_distribution_file_parsed": true,
"ansible_distribution_file_path": "/etc/redhat-release",
"ansible_distribution_file_variety": "RedHat",
"ansible_distribution_major_version": "7",
"ansible_distribution_release": "Core",
"ansible_distribution_version": "7.5.1804",
"ansible_dns": {
  "nameservers": [
    "127.0.0.1"
  ]
},
"ansible_domain": "",
"ansible_effective_group_id": 1000,
"ansible_effective_user_id": 1000,
"ansible_env": {
  "HOME": "/home/zuul",
  "LANG": "en_US.UTF-8",
  "LESSOPEN": "||/usr/bin/lesspipe.sh %s",
  "LOGNAME": "zuul",
  "MAIL": "/var/mail/zuul",
  "PATH": "/usr/local/bin:/usr/bin",
  "PWD": "/home/zuul",
  "SELINUX_LEVEL_REQUESTED": "",
  "SELINUX_ROLE_REQUESTED": "",
  "SELINUX_USE_CURRENT_RANGE": "",
  "SHELL": "/bin/bash",
  "SHLVL": "2",
  "SSH_CLIENT": "REDACTED 55672 22",
  "SSH_CONNECTION": "REDACTED 55672 REDACTED 22",
  "USER": "zuul",
  "XDG_RUNTIME_DIR": "/run/user/1000",
  "XDG_SESSION_ID": "1",
  "_": "/usr/bin/python2"
},
"ansible_eth0": {
  "active": true,
  "device": "eth0",
  "ipv4": {
    "address": "REDACTED"
```

Anatomy of a playbook

'-' at the lowest level of indentation represent

plays

hosts apply **plays** to host groups defined in the inventory file

becomes determines privilege escalation

tasks are defined for each **play**

each **task** has a **name**

each **task** has a **module** that defines what the task does

```
---
- name: Configure webserver on the server instance
  hosts: appserver
  connection: ssh
  become: true

  tasks:
    # Create and Setup Nginx
    - name: Copy nginx config files
      copy:
        src: "./nginx-conf/nginx"
        dest: "/conf"
    - name: Create nginx container
      docker_container:
        name: "nginx"
        image: "nginx:stable"
        state: started
        recreate: yes
        published_ports:
          - 80:80
          - 443:443
        networks:
          - name: "{{docker_network_name}}"
        volumes:
          - /conf/nginx/nginx.conf:/etc/nginx/nginx.conf

    - name: "Restart nginx container"
      shell: "docker container restart nginx"
```

Plays

Tasks

Modules

Deployment Steps (Ansible / Automation)

1. Setup local container to connect to GCP
2. Build and push docker images to GCR
3. Create Compute Instance (VM) in GCP
4. Provision the server (Installed required softwares)
5. Setup Docker containers in VM Instance
6. Setup a web server to expose our app to the outside world

Setup local container /GCP

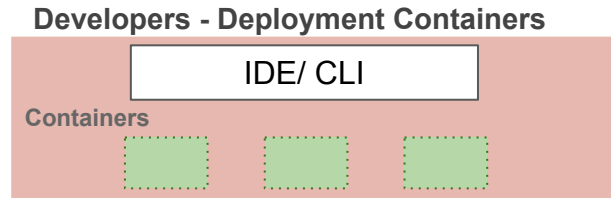
Setup required GCP

- Enable APIs
- Create service accounts
 - **deployment** (To deploy everything to GCP)
 - **gcp-service** (To read containers from GCR in VM)

Setup local deployment container

- Add secret keys
- Set GCP project we want to connect to

Build & Push Docker Images to GCR



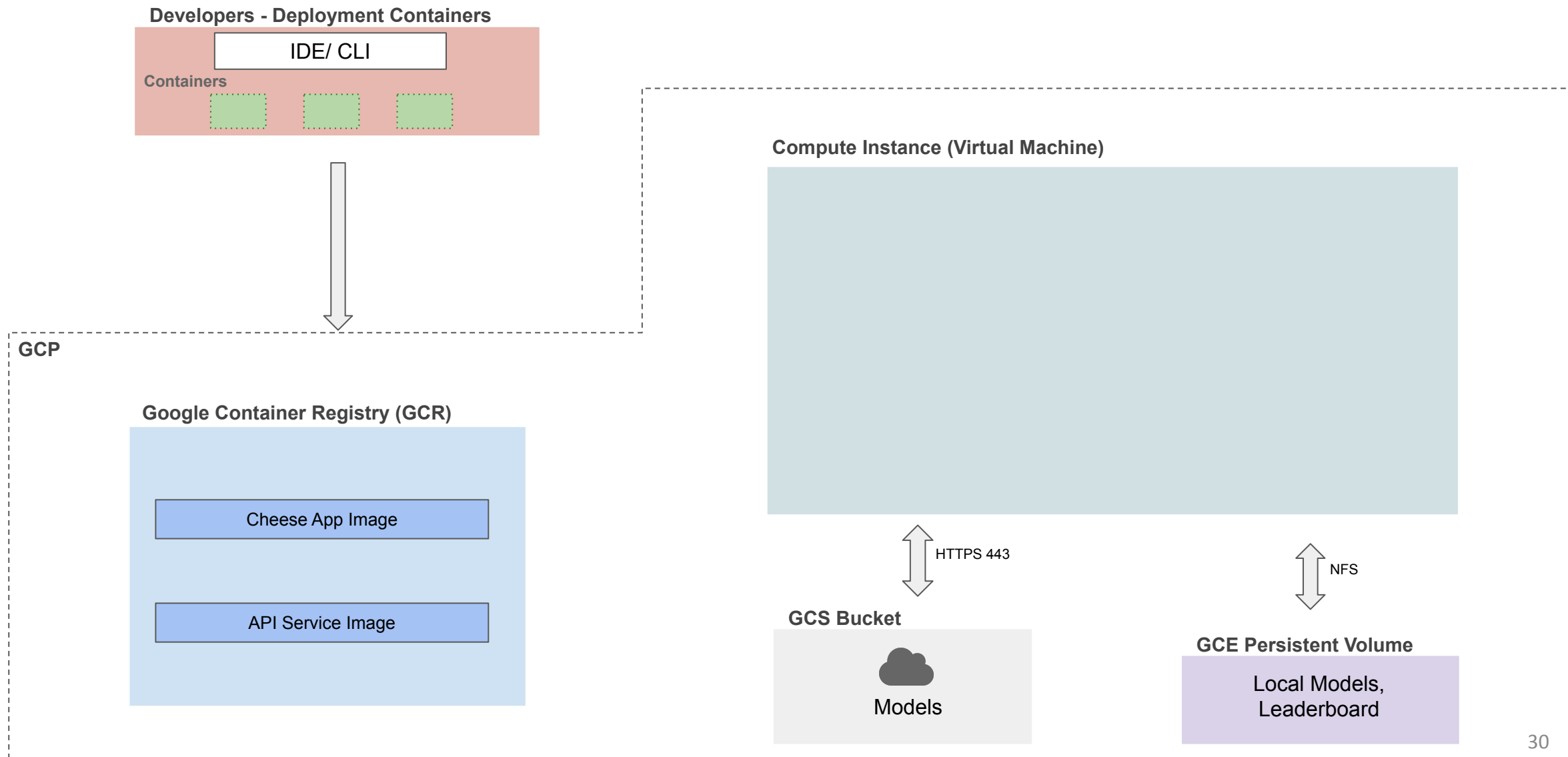
GCP

Google Container Registry (GCR)

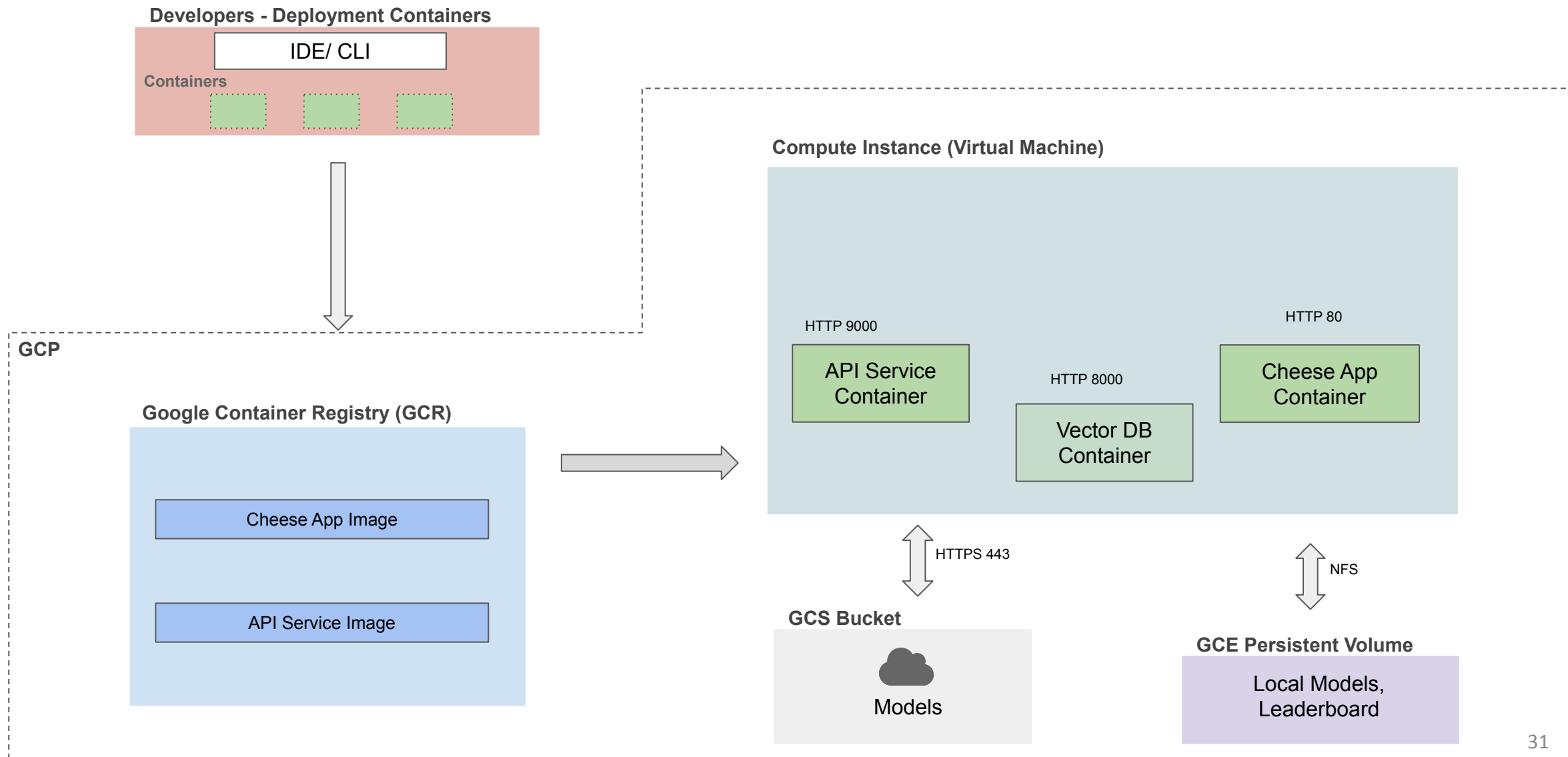
Cheese App Image

API Service Image

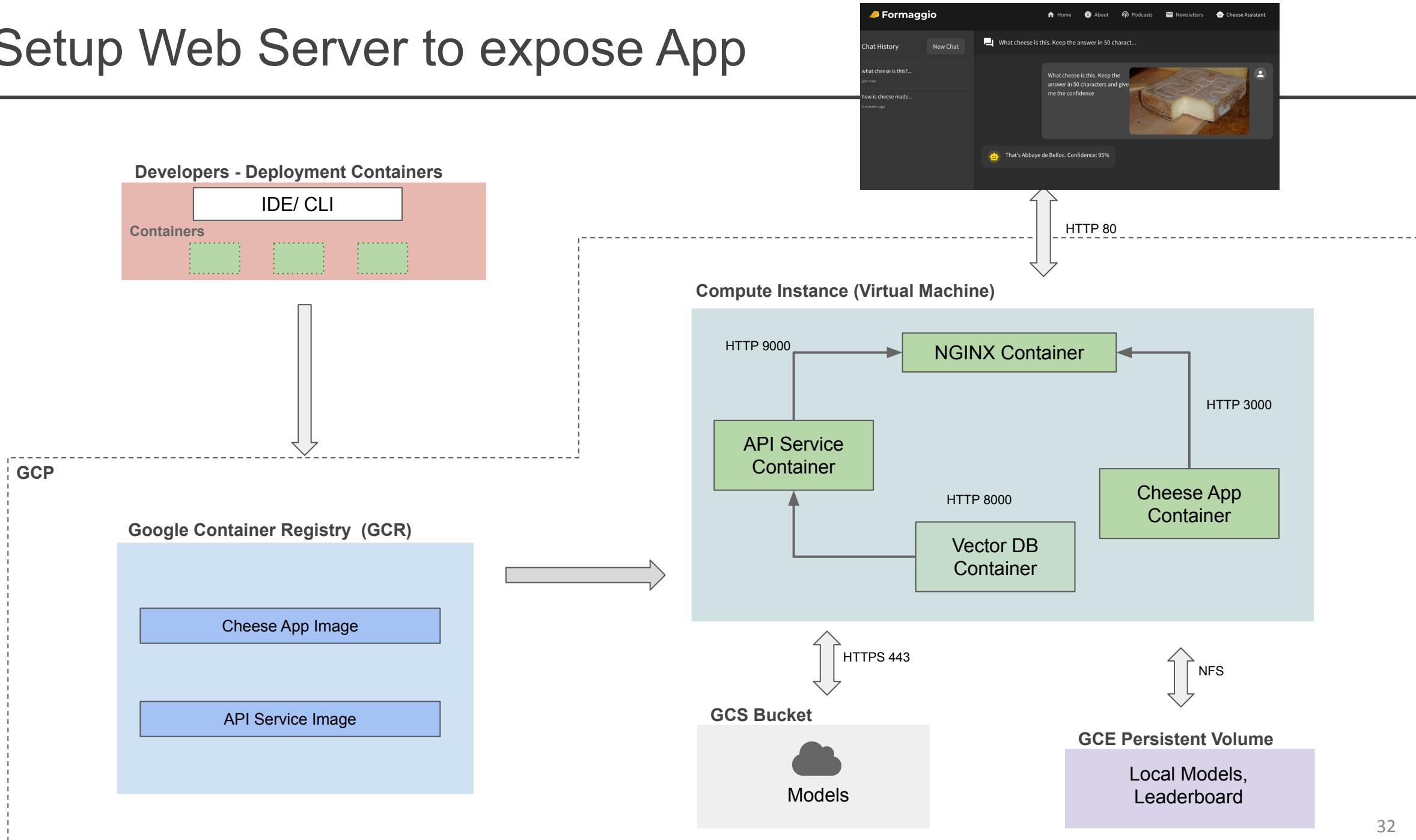
Create Compute Instance (VM)



Setup Docker Containers in VM



Setup Web Server to expose App

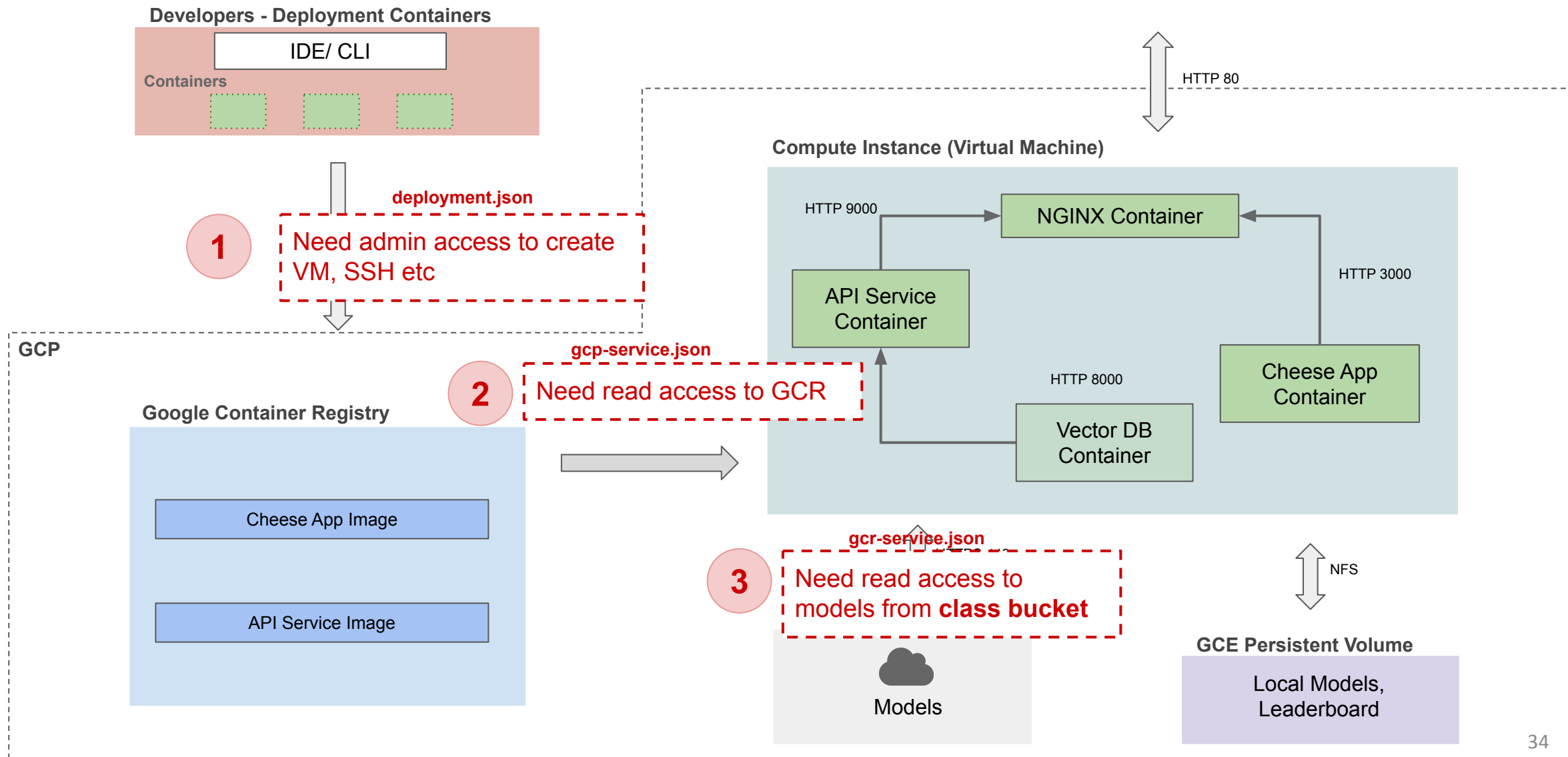


Why did we need 2 service accounts?

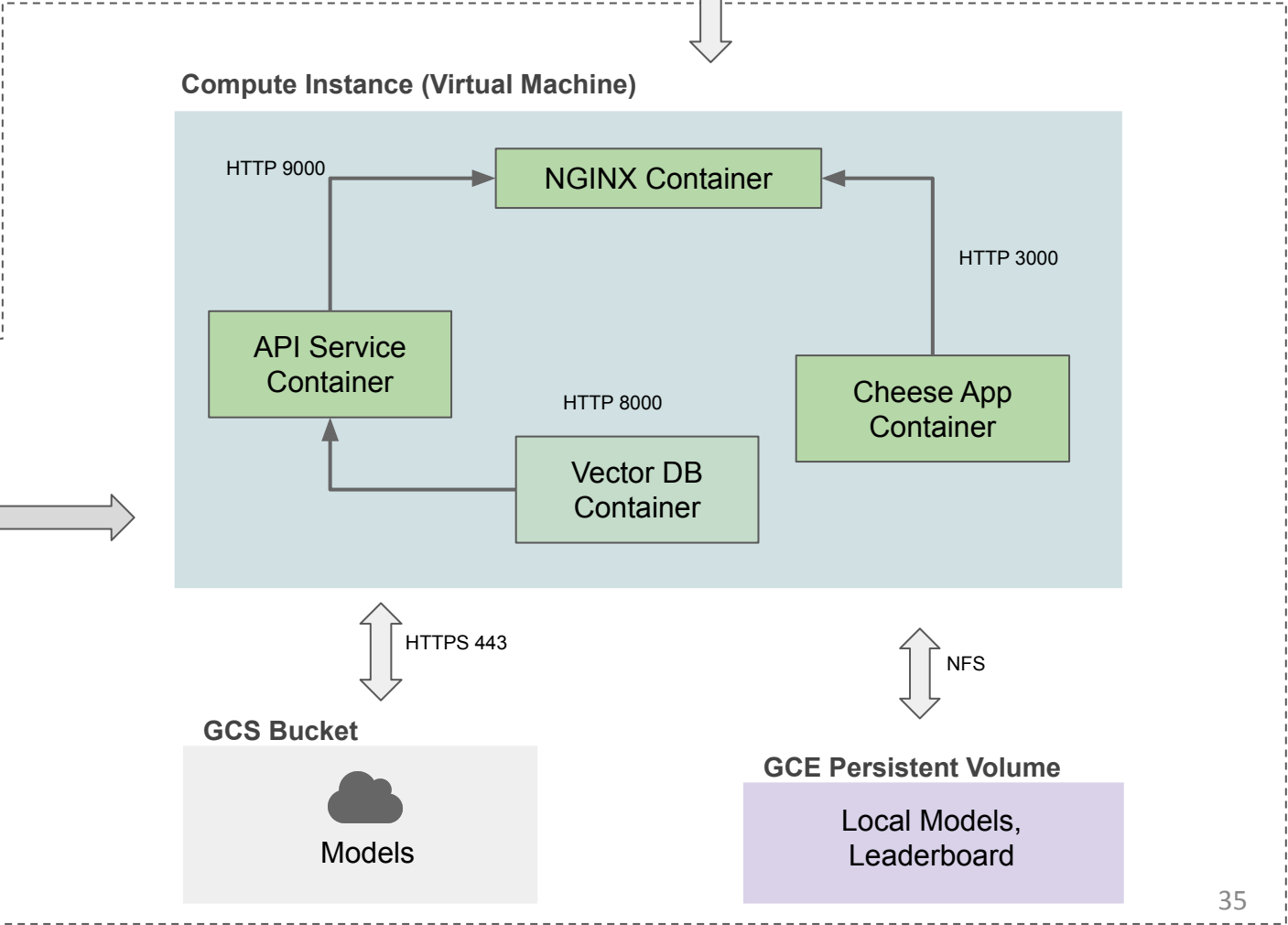
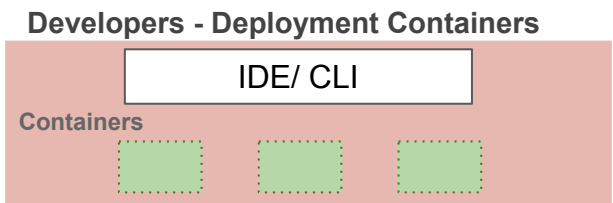
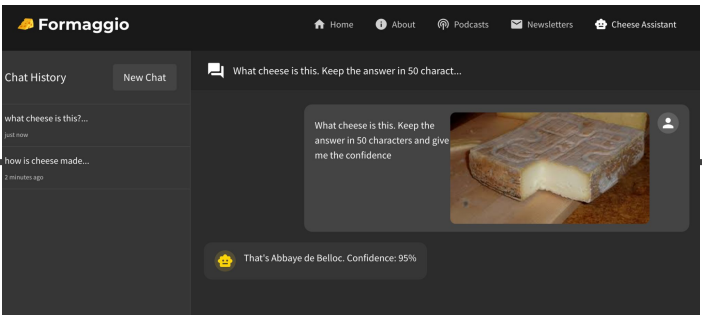
Why do we need the following service accounts?

- **deployment**
 - Has admin access to **your** group GCP project
- **gcp-service**
 - Has read access to **your** group GCP projects GCR
 - Has access to Vertex AI to perform inference

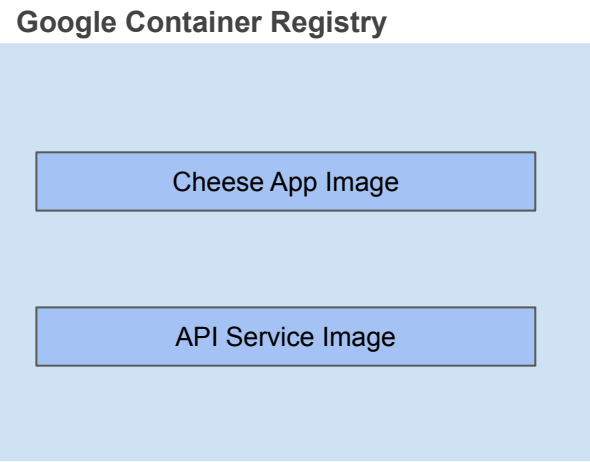
Why did we need 2 service accounts?



Test the App



GCP



Tutorial: Deployment to GCP (Ansible)

[Cheese App - Deployment to GCP \(Ansible\)](#)

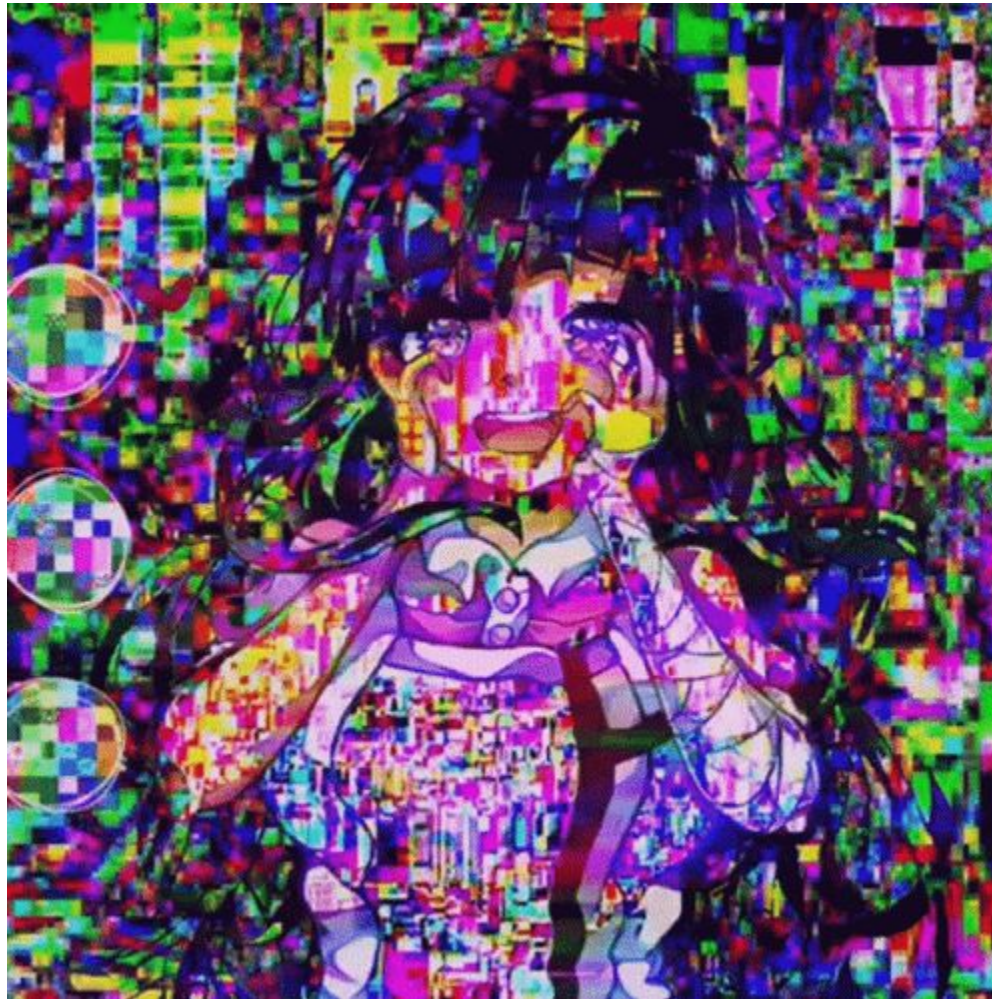
Build & Push Images: Use `deploy-docker-images.yml` to build and push Docker containers to Google Container Registry (GCR)

Create VM Instance: Deploy a Compute Engine VM using `deploy-create-instance.yml` with the specified configuration in `inventory.yml`

Provision Instance: Install required dependencies and setup environment using `deploy-provision-instance.yml`

Deploy Application: Configure and launch Docker containers (API, Frontend, Nginx) using `deploy-setup-containers.yml` and `deploy-setup-webserver.yml`





DON'T FORGET TO DELETE

THANK YOU!!!

