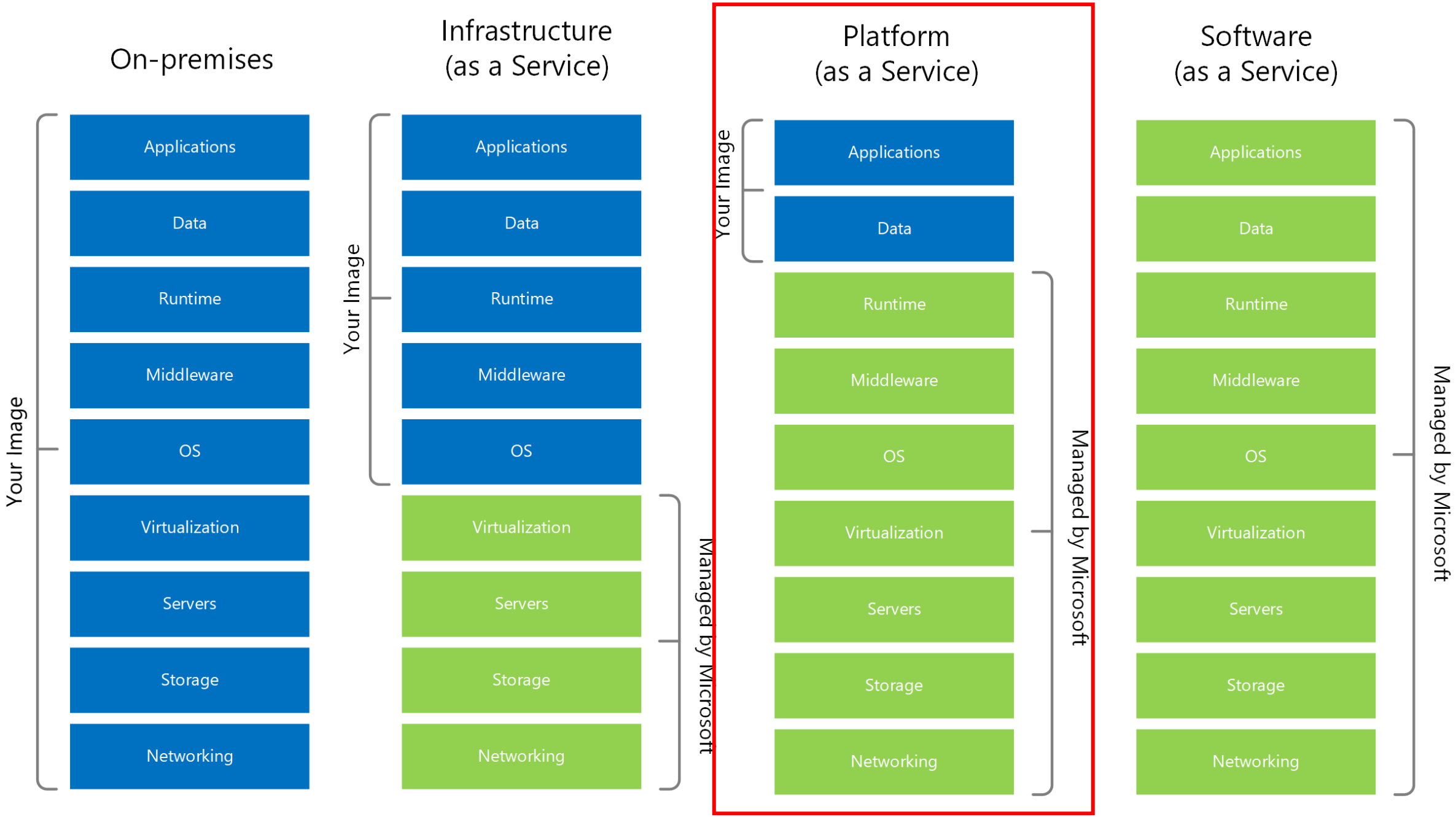
App Engine

Memi Lavi
www.memilavi.com

App Engine

- Fully managed runtime platform
- Runs your code
- Scales automatically
- Takes care of all infrastructure aspects
- All you have to do is upload your code


On-premises
Infrastructure (as a Service)
Platform (as a Service)
Software (as a Service)
Your Image
Applications
Data
Runtime
Middleware
OS
Virtualization
Servers
Storage
Networking
Managed by Microsoft

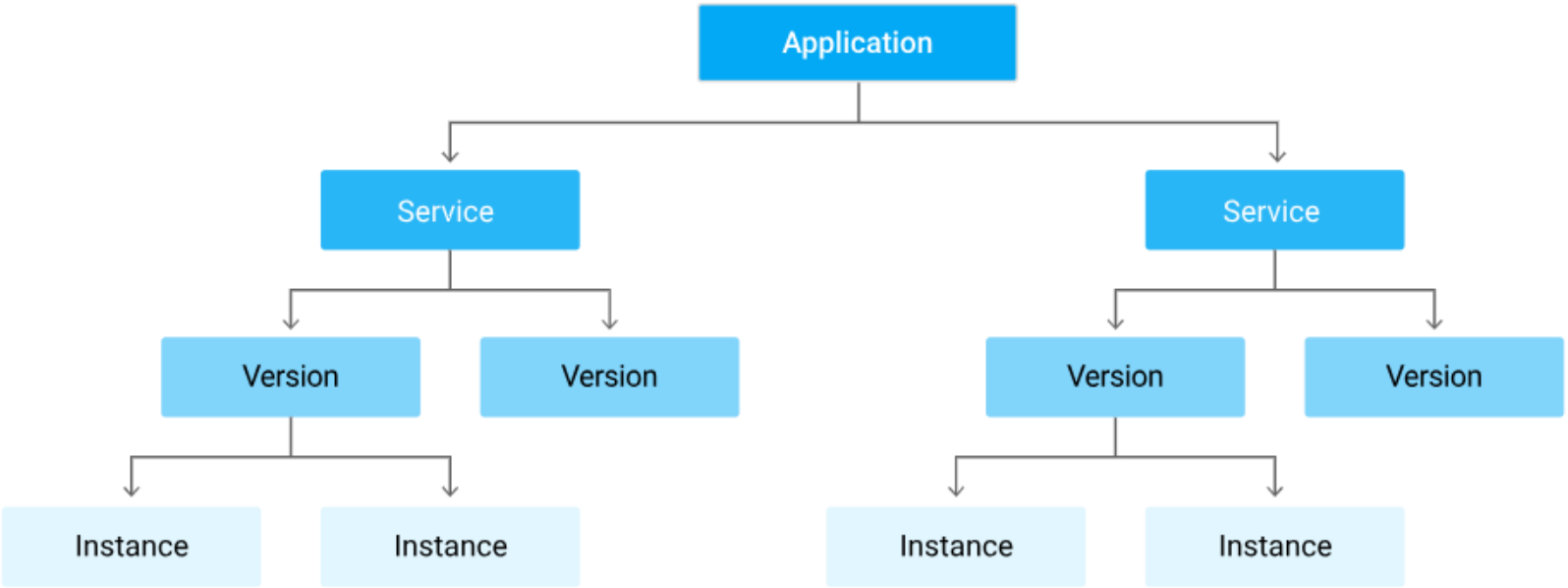
App Engine Components

- One App Engine per project
- In a single region
- Code deployed into Services
- As many services as needed
- Services can communicate between them
- Great for Microservices systems

App Engine Components

- Service can have multiple versions
- Traffic can be split between versions
- Versions run on instances
  - The actual compute resources
- Instances are scaled as needed
  - Depends on configuration

App Engine Components



Source: https://cloud.google.com/appengine/docs/an-overview-of-app-engine

App Engine Components

- When App Engine is enabled a service named `default` is created
- Other services can be created as needed
- All in the same region
- App Engine SLA: 99.95%
- App Engine pricing: Depends on App Engine Type

App Engine Types

Standard

- Runs in a sandbox using lightweight instance
- Supports specific versions of specific languages
- Quick startup and scaling
- No background process
- Limited writing to disk
- Can be scaled to 0 instances
- Quick deployment time (seconds)
- Complex local development (requires SDK)
- No Websockets support
- Pricing: Based on Instance hours

Flexible

- Runs in docker container on regional MIG
- Supports almost every version of every runtime
- Takes time to start and scale
- Supports background process
- Supports writing to disk
- Minimum 1 instance
- Slower deployment time (minutes)
- Simple local development (container based)
- Websockets support
- Pricing: Based on vCPU, memory and persistent disks

App Engine Types

Standard

- Applications that require rapid scaling
- Free-to-low cost
- Can do with limited compute resources

When to use?

Flexible

- Applications with consistent traffic with regular fluctuations
- Flexible compute resources

Standard Environment

- Platform support:
  - Python 2.7-3.11
  - Java 8,11,17
  - Node.js 10,12,16,18,20
  - PHP 7.2-8.2
  - Go 1.12-1.21 (preview)

Standard Environment

- Autoscaling support:

Automatic (default)	Based on request rate, latency and other metrics. Can set minimum number of instances
Basic	Based on received requests. Instances shut down when go idle
Manual	Manually specify the number of running instances

Standard Environment

- Instance Classes:

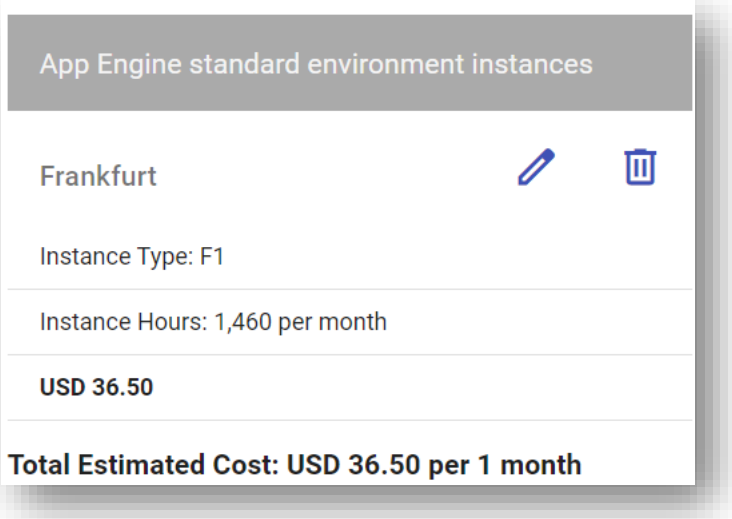
Instance Class	Memory Limit	CPU Limit	Supported Scaling Types
F1 (default)	384 MB	600 MHz	automatic
F2	768 MB	1.2 GHz	automatic
F4	1536 MB	2.4 GHz	automatic
F4_1G	3072 MB	2.4 GHz	automatic
B1	384 MB	600 MHz	manual, basic
B2 (default)	768 MB	1.2 GHz	manual, basic
B4	1536 MB	2.4 GHz	manual, basic
B4_1G	3072 MB	2.4 GHz	manual, basic
B8	3072 MB	4.8 GHz	manual, basic

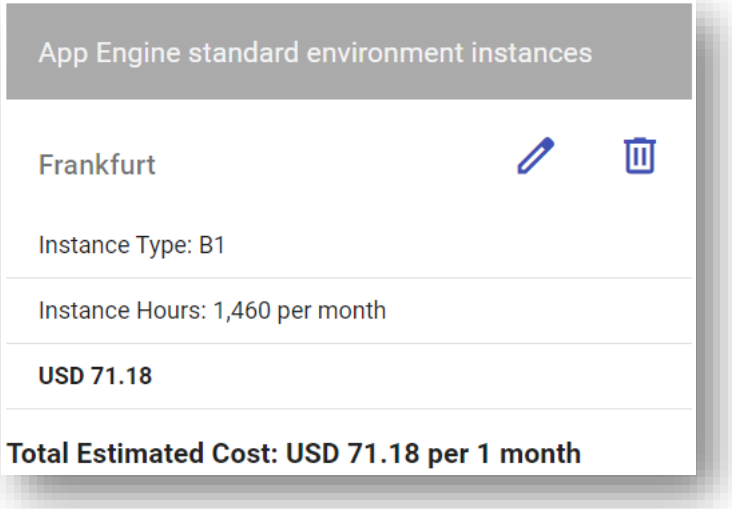
Standard Environment

- Pricing:
  - Free tier:
    - “F” instances: 28 instance hrs / day
    - “B” instances: 9 instance hrs / day

Standard Environment

- Pricing:
  - After that:





Flexible Environment

- Platform support:
  - Python, Java, Node.js, Go, Ruby, PHP, .NET
  - Custom runtime in other languages

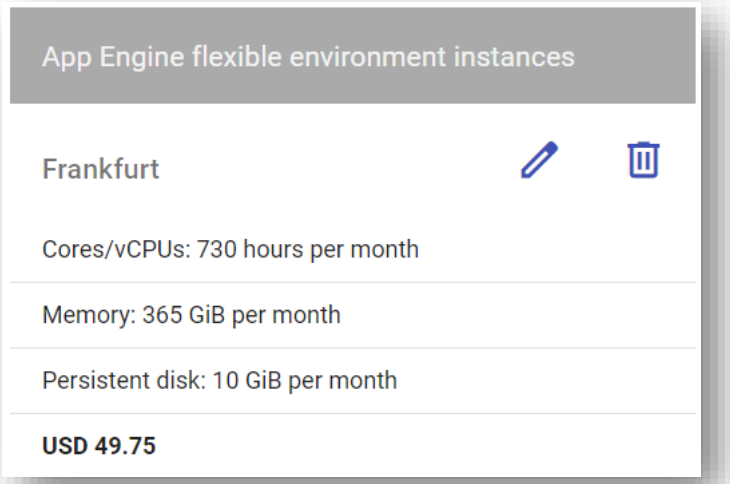
Flexible Environment

- Autoscaling support:

Automatic (default)	Based on request rate, latency and other metrics. Can set minimum number of instances
Manual	Manually specify the number of running instances

Flexible Environment

- Pricing:



Deploying App Engine Service

Create App

Create the App Engine Application which will be used in the project.

Configure

Add `app.yaml` configuration file to the code, containing the configuration of the service.

Deploy

Deploy the service using `gcloud`, specify the version number.

Create App

- App Engine runs in an application
- Services are deployed in the application
- One per project
- Can be done from the Console, gcloud, or API

Configure

- app.yaml defines the configuration of the App Engine's service
- Sets, among others:
  - Environment type (standard vs flexible)
  - Runtime (.NET, Java, PHP etc.)
  - Scaling
  - Health checks
  - Networking
  - Environment variables
  - More…

Deploy

- Use the `gcloud` CLI to deploy the service
- From the cloud or locally
- Important: Specify version #
  - Helps in splitting traffic, managing audience and more
  - We'll talk about it later in this section

App Engine URLs

- Default service:
  - `<PROJECT_ID>.<REGION_ID>.r.appspot.com`
  - Example: `marine-resource-395306.ey.r.appspot.com`
- Other services:
  - `<SERVICE>-dot-<PROJECT_ID>.<REGION_ID>.r.appspot.com`
  - Example: `inventory-dot-marine-resource-395306.ey.r.appspot.com`

Cloud Build

- CI\CD service in the Google Cloud
- Builds source code and deploys it in the cloud
- Integrates with:
  - App Engine
  - Cloud Run
  - Google Kubernetes Engine
  - Cloud Functions
  - Firebase

Cloud Build

- Can be used to continuously deploy code to App Engine
- Triggered automatically or manually
- Supports pipelines with multiple build and test steps
- Used when deploying code
- We'll use it in the Flexible environment deployment demo

App Engine Versions

- Service can have multiple versions
- By default, all traffic is routed to the new version
- Previous versions can still be accessed:
  - `<VERSION>-dot-<SERVICE>-dot-<PROJECT_ID>.<REGION_ID>.r.appspot.com`
  - Example: `v1-inventory-dot-marine-resource-395306.ey.r.appspot.com`

Traffic Splitting

- Traffic between versions can be customized
- Specific percentage of traffic will go to each version
- Great for gradual updates
- Can be set from the console or as part of the deployment

Deployment Types

- Traffic Splitting enables various types of deployment

Basic

Rolling

Blue-Green

Canary

Basic Deployment

- All instances update to the new version at once

v1

v2

Basic Deployment

- All instances update to the new version at once

v1

v2

Basic Deployment

Pros

- Simple
- Fast

Cons

- Risky
- System might get unusable

Basic Deployment

- App Engine implementation:
  - Simple deployment of new version
  - All traffic automatically routed to the new version
  - The previous version is shut down

Rolling Deployment

- Instances are updated gradually in batches
- Only if no errors are found the deployment resumes

v1

v2

Rolling Deployment

- Instances are updated gradually in batches
- Only if no errors are found the deployment resumes

v1

v2

Rolling Deployment

- Instances are updated gradually in batches
- Only if no errors are found the deployment resumes

v1

v2

Rolling Deployment

- Instances are updated gradually in batches
- Only if no errors are found the deployment resumes

v1

v2

Rolling Deployment

Pros

- Allows rollback

Cons

- Need to support two versions simultaneously
- Not easy to manage

Rolling Deployment

- App Engine implementation:
  - After deployment of new version split traffic so that new version gets small % of traffic
  - Set Split Type to Random
  - Gradually increase % of traffic to new version until 100%

Blue-Green Deployment

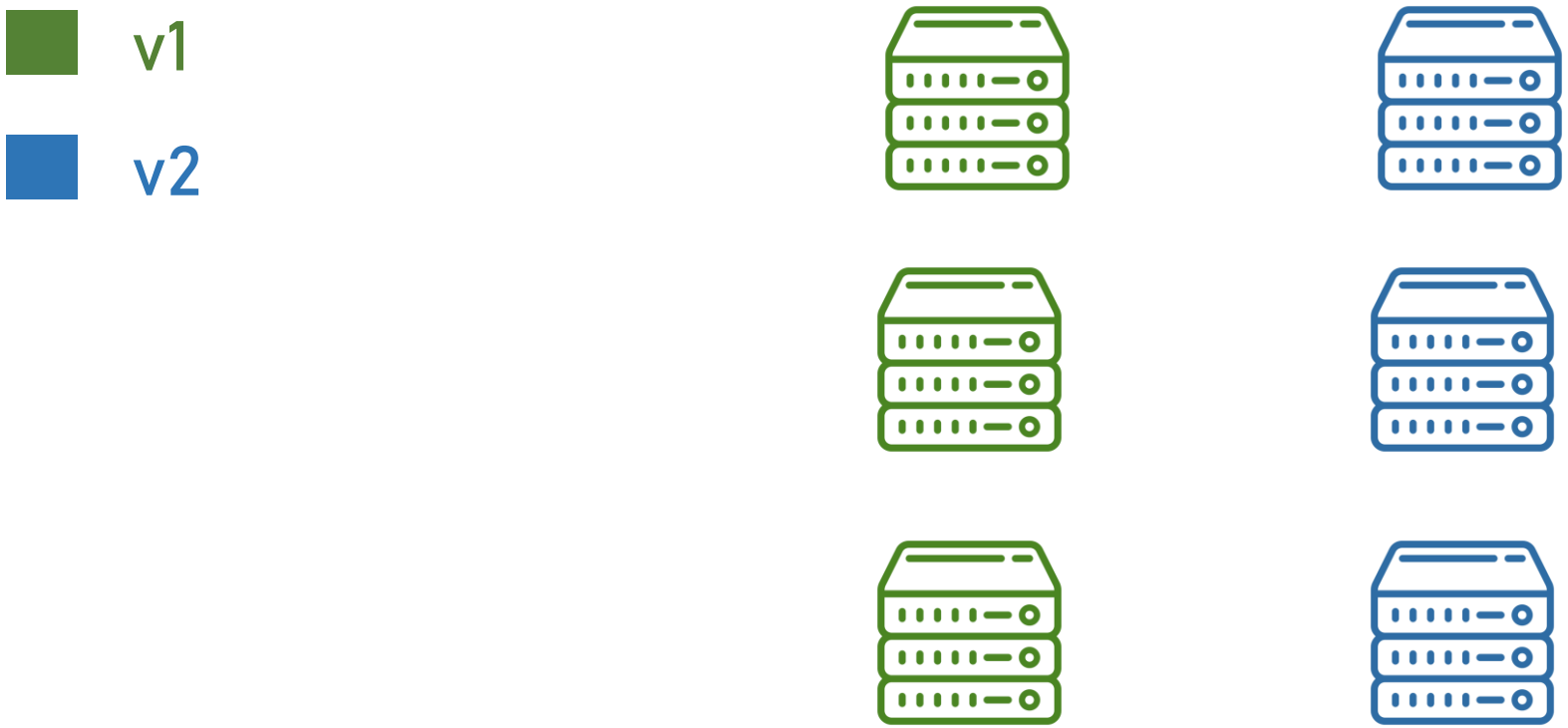
- New version uploaded and accessible only to testers
- After verification complete, traffic is routed to new version

v1

v2

Blue-Green Deployment

- New version uploaded and accessible only to testers
- After verification complete, traffic is routed to new version



Blue-Green Deployment

- New version uploaded and accessible only to testers
- After verification complete, traffic is routed to new version

v1

v2

Blue-Green Deployment

Pros

- Simple
- New environment is always tested

Cons

- Cost
  - More instances

Blue-Green Deployment

- App Engine implementation:
  - After deployment of new version allocate 100% of traffic to the previous version
  - Testers work on the new version using dedicated URL
    - **<VERSION>-**dot-<SERVICE>-dot-<PROJECT_ID>.<REGION_ID>.r.appspot.com
  - After testing is complete – 100% of traffic is routed to the new version

Canary Deployment

- Instances are updated gradually in batches
- Only if no errors are found by specific testers deployment resumes

v1

v2

Canary Deployment

- Instances are updated gradually in batches
- Only if no errors are found by specific testers deployment resumes

v1

v2

Canary Deployment

- Instances are updated gradually in batches
- Only if no errors are found by specific testers deployment resumes

v1

v2

Canary Deployment

- Instances are updated gradually in batches
- Only if no errors are found by specific testers deployment resumes

v1

v2

Canary Deployment

Pros

- Allows rollback
- Controlled

Cons

- Need to support two versions simultaneously
- Not easy to manage

Canary Deployment

- App Engine implementation:
  - After deployment of new version split traffic so that new version gets small % of traffic
  - Set Split Type to IP Address
  - Gradually increase % of traffic to new version until 100%

Scheduling App Engine

- App Engine uses the request / response model
- Users call it and get a response
- Sometimes it's required to schedule calls to App Engine without user intervention
  - Running batches, retrieving data from other systems, etc.

Scheduling App Engine

- Two mechanisms for scheduling calls to App Engine:

cron job

- Specified using `cron.yaml` file
- Runs GET calls only to the App Engine
- Simple scheduling support
- Free

Cloud Scheduler

- Separate resource
- Can call other apps
- Complex scheduling support
- Not free

cron.yaml

```yaml
cron:
- description: "daily summary job"
  url: /tasks/summary
  schedule: every 24 hours
- description: "monday morning mailout"
  url: /mail/weekly
  schedule: every monday 09:00
  timezone: Australia/NSW
- description: "new daily summary job"
  url: /tasks/summary
  schedule: every 24 hours
  target: beta
```

service name

Cloud Scheduler cost

- Pricing based on Jobs
  - Definition of schedule and execution
- $0.10 / job / month
- First three jobs are free (per account)

Cloud Scheduler cost

Cloud Scheduler

Total amount of jobs: 5

USD 0.20

Total Estimated Cost: USD 0.20 per 1 month

Estimate Currency

USD - US Dollar

ReadIt!

Architecture: ReadIt Cloud System