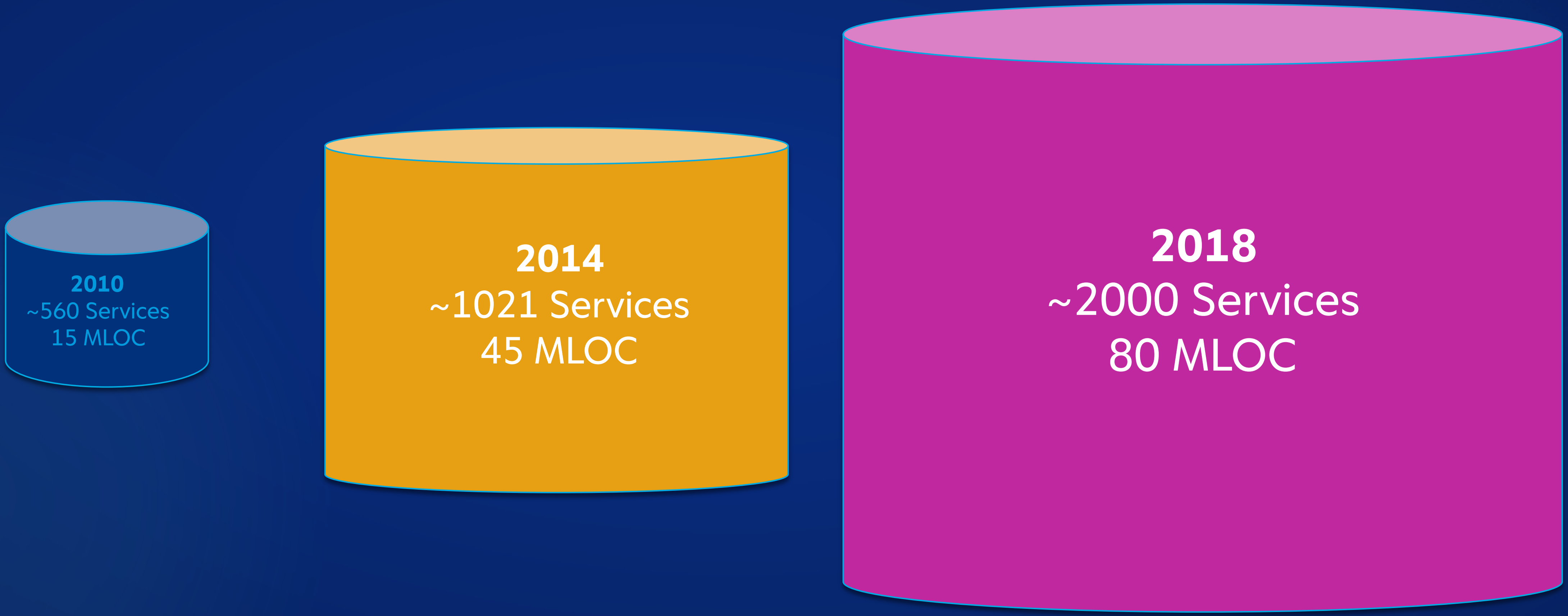
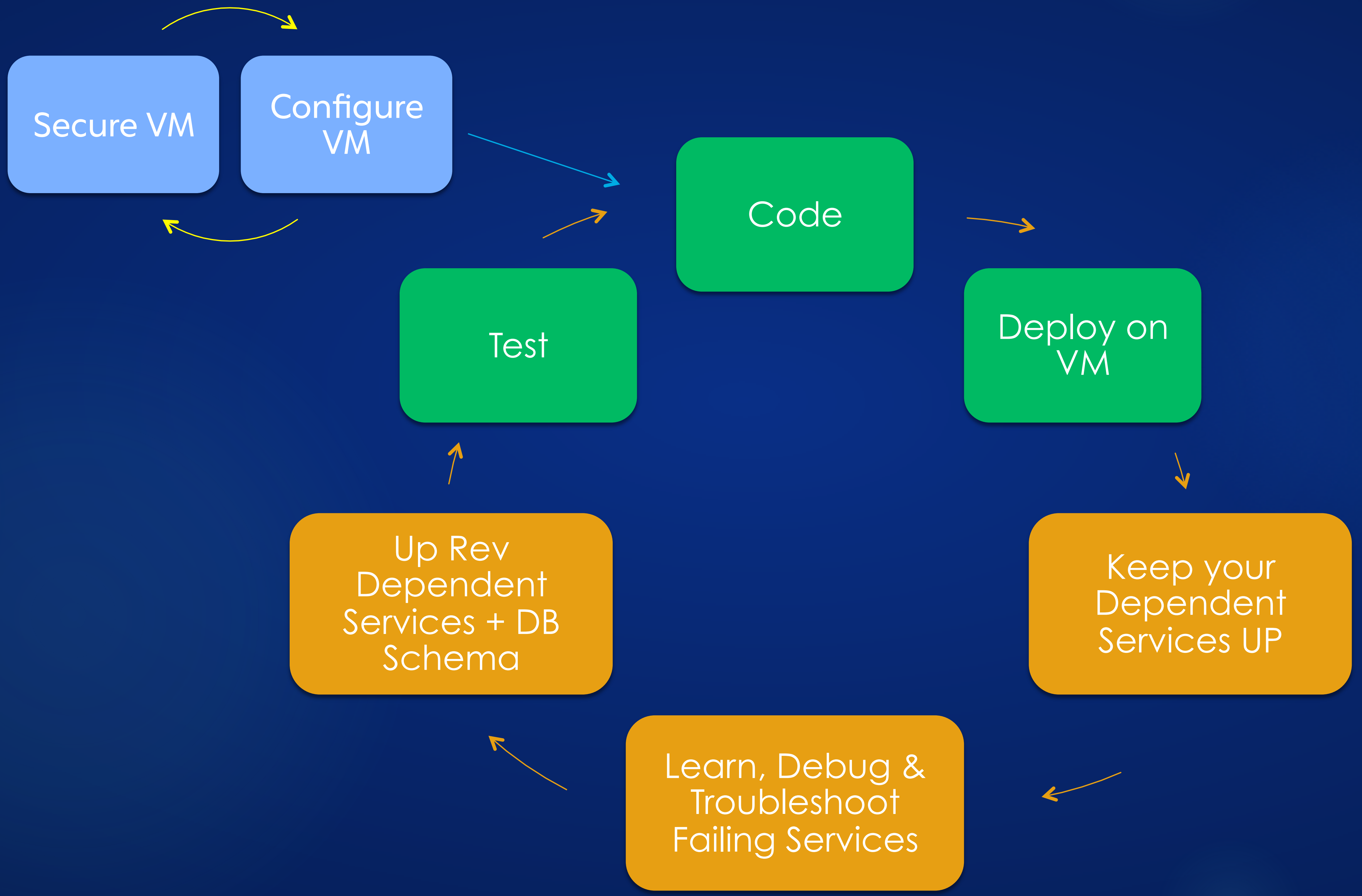
Save Millions By Efficient Resource Utilization Through Mesos

▶By Smarth Madan

PayPal and its Code is growing YOY <WIP>



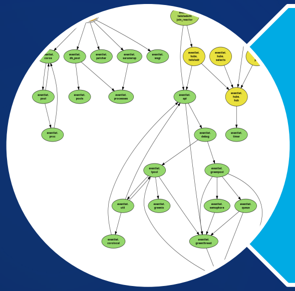
Test Cycle at PayPal Before Managed Stage


Secure VM
Configure VM
Code
Deploy on VM
Keep your Dependent Services UP
Learn, Debug & Troubleshoot Failing Services
Up Rev Dependent Services + DB Schema
Test

Developer Pain points

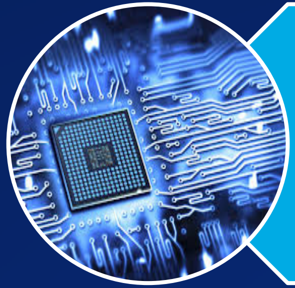
Deploying all the components

Tons of time for setup and test


Identifying transitive dependencies

Maintaining stable environment

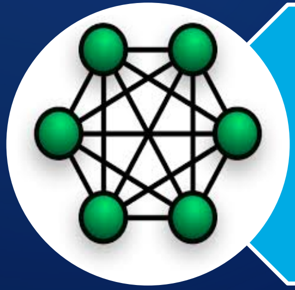
Infrastructure Team Pain points



Hardware requirements grows YOY

Huge Maintenance cost for ~4K test Environment

Network Bandwidth



Test topology is not same prod

Requirements

- We need Production Like environment
- Cluster of machines running all services
- Multiple instances of each service with auto healing for availability
- Needed to scale as the number of users grow
- Code refresh in mins
- Easy to connect from all other VMs

Resource Management

- Manage a large set of machines in terms of compute
- Moving away from static allocation of machines
- Identifying unused capacity
- Slave Categorization (diverse PayPal Tech stack)

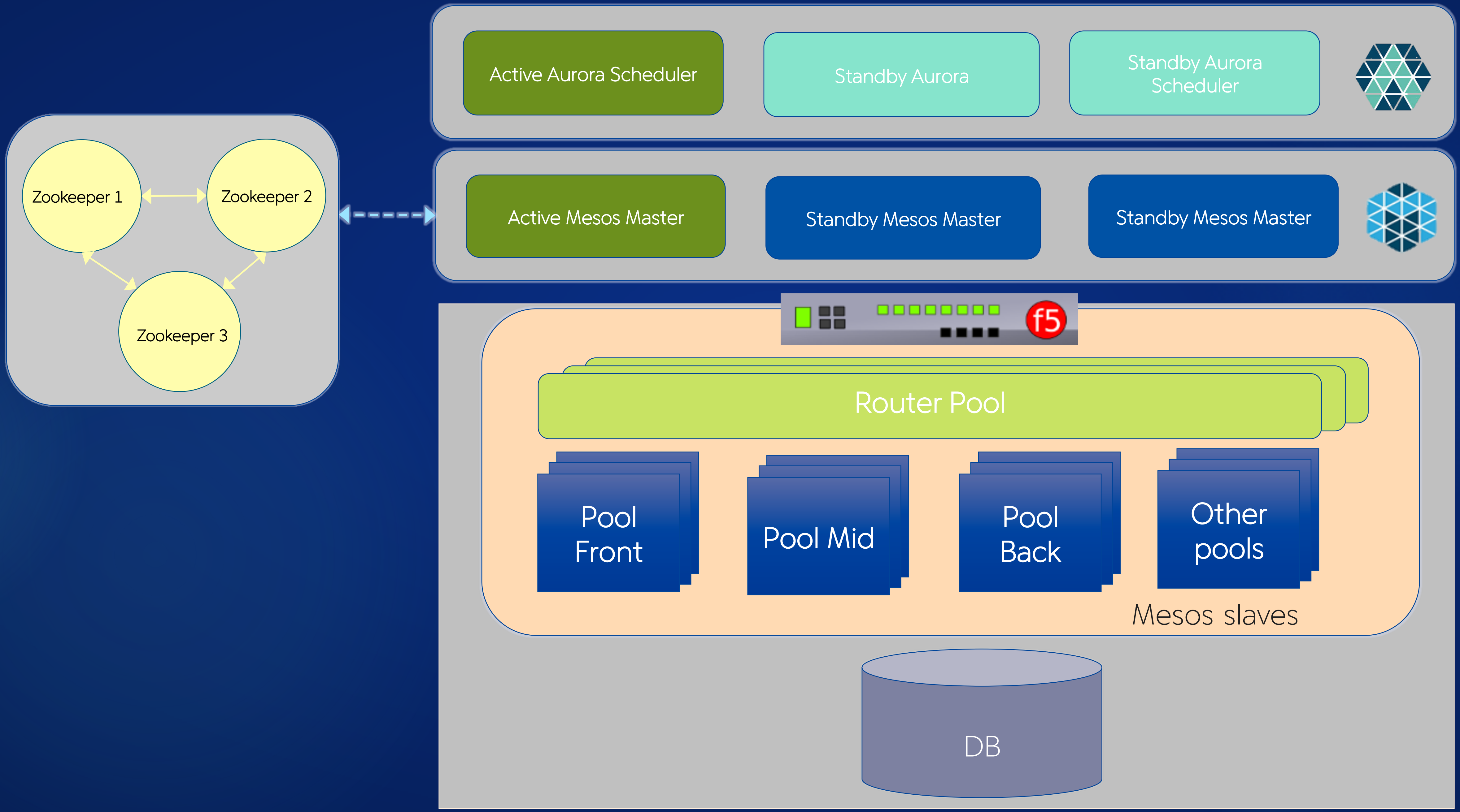
Job Scheduler

- Defines jobs with ordered tasks
- Binding jobs to specific salves using constraints
- Auto-healing
- REST API capability
- Final task to cleanup the slave for any new job

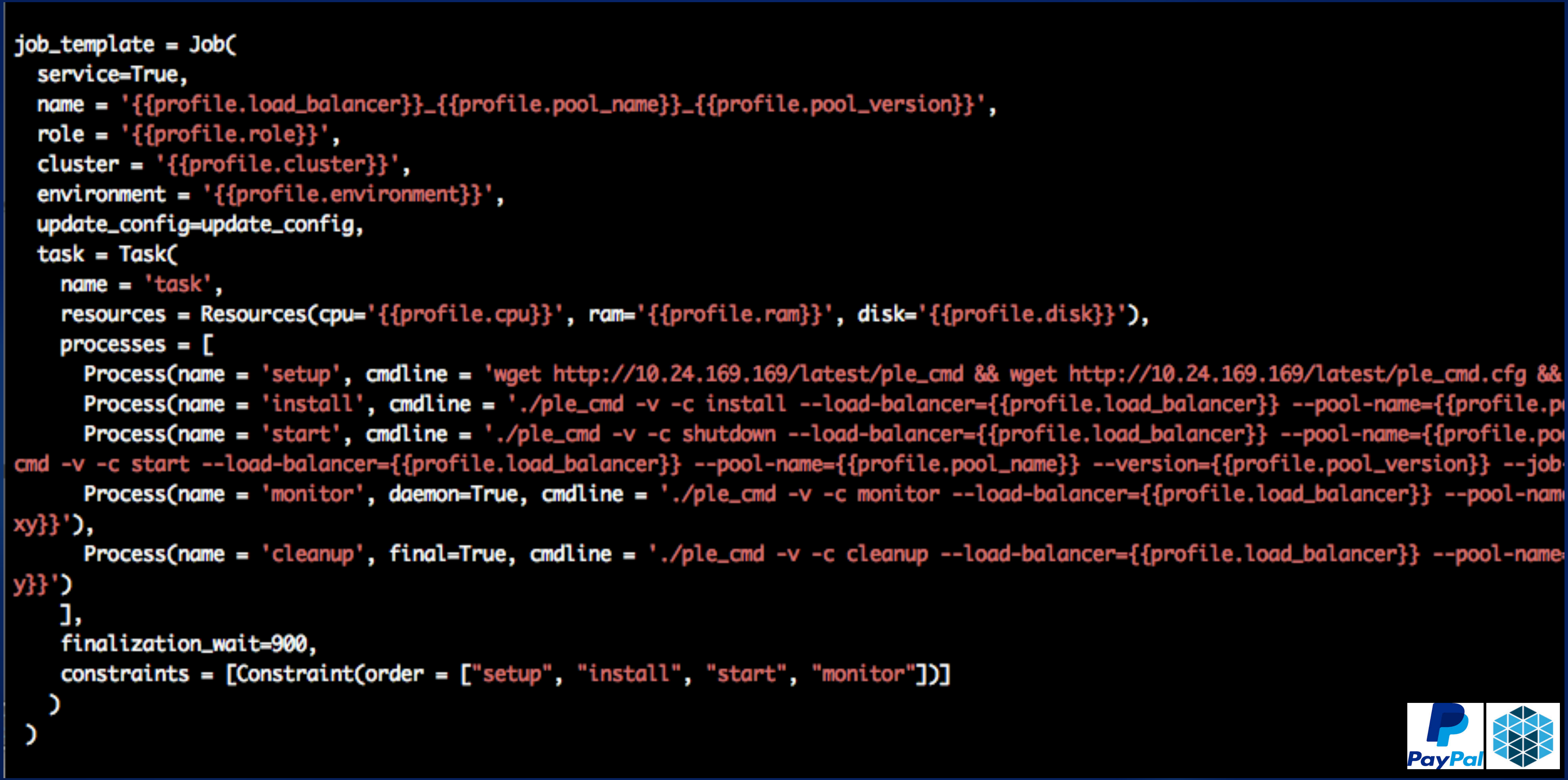
What’s a Managed Stage?

Managed Stage is a continuously available Mesos based multi-node staging environment that allows Developers and Quality engineers to certify and release apps to LIVE.

Managed Stage Architecture



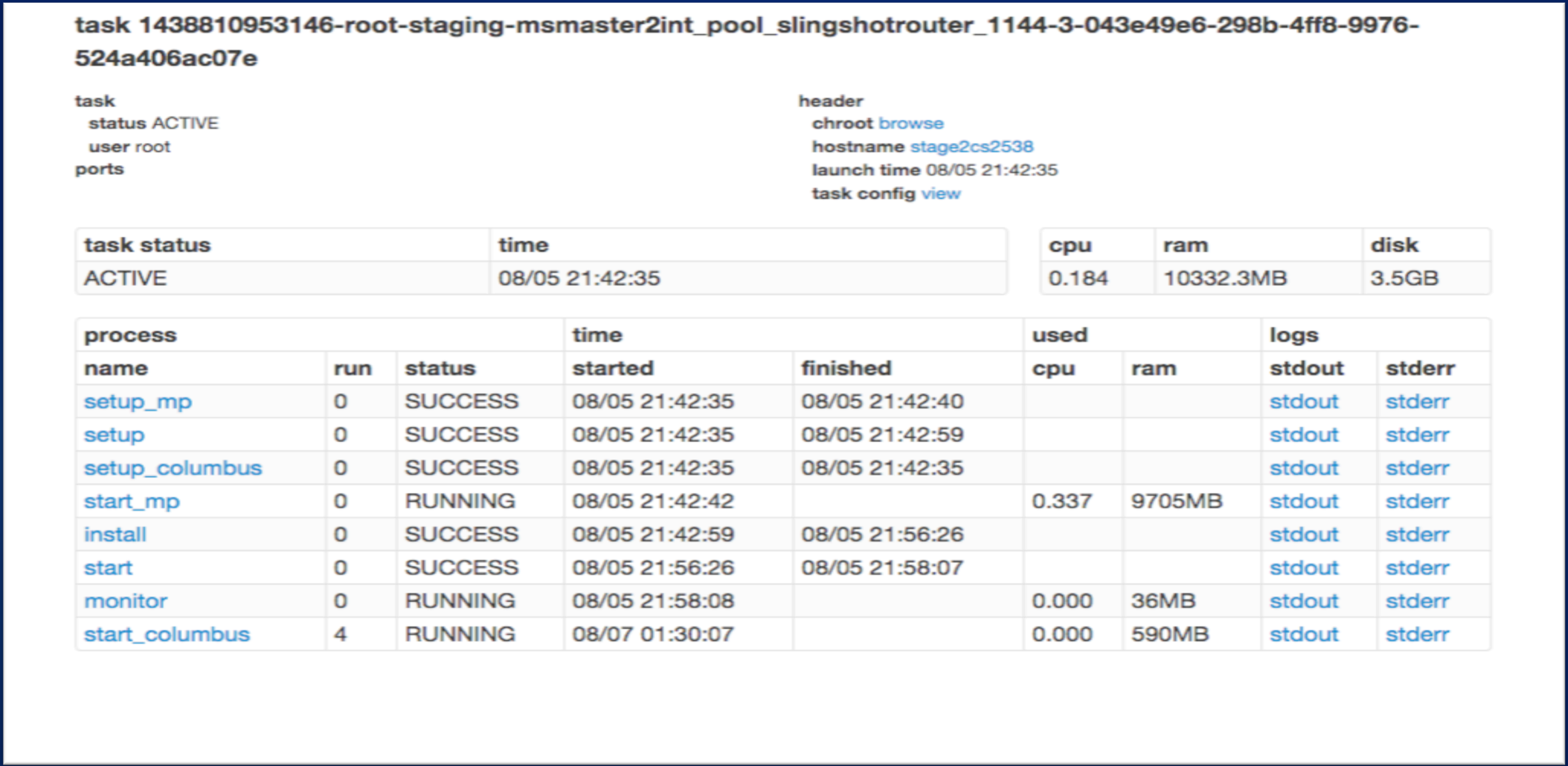
Sample Aurora Task



```
job_template = Job(
  service=True,
  name = '{{profile.load_balancer}}_{{profile.pool_name}}_{{profile.pool_version}}',
  role = '{{profile.role}}',
  cluster = '{{profile.cluster}}',
  environment = '{{profile.environment}}',
  update_config=update_config,
  task = Task(
    name = 'task',
    resources = Resources(cpu='{{profile.cpu}}', ram='{{profile.ram}}', disk='{{profile.disk}}'),
    processes = [
      Process(name = 'setup', cmdline = 'wget http://10.24.169.169/latest/ple_cmd && wget http://10.24.169.169/latest/ple_cmd.cfg &&
      Process(name = 'install', cmdline = './ple_cmd -v -c install --load-balancer={{profile.load_balancer}} --pool-name={{profile.p
      Process(name = 'start', cmdline = './ple_cmd -v -c shutdown --load-balancer={{profile.load_balancer}} --pool-name={{profile.po
cmd -v -c start --load-balancer={{profile.load_balancer}} --pool-name={{profile.pool_name}} --version={{profile.pool_version}} --job
      Process(name = 'monitor', daemon=True, cmdline = './ple_cmd -v -c monitor --load-balancer={{profile.load_balancer}} --pool-nam
xy}}'),
      Process(name = 'cleanup', final=True, cmdline = './ple_cmd -v -c cleanup --load-balancer={{profile.load_balancer}} --pool-name
y}}')
    ],
    finalization_wait=900,
    constraints = [Constraint(order = ["setup", "install", "start", "monitor"])]
  )
 )
```

Complex Job Scenario

```
constraints = order("setup", "install", "start","monitor")+order("setup_columbus","start_columbus")+order("setup_mp","start_mp")
```



task 1438810953146-root-staging-msmaster2int_pool_slingshotrouter_1144-3-043e49e6-298b-4ff8-9976-524a406ac07e

task
  status ACTIVE
  user root
ports

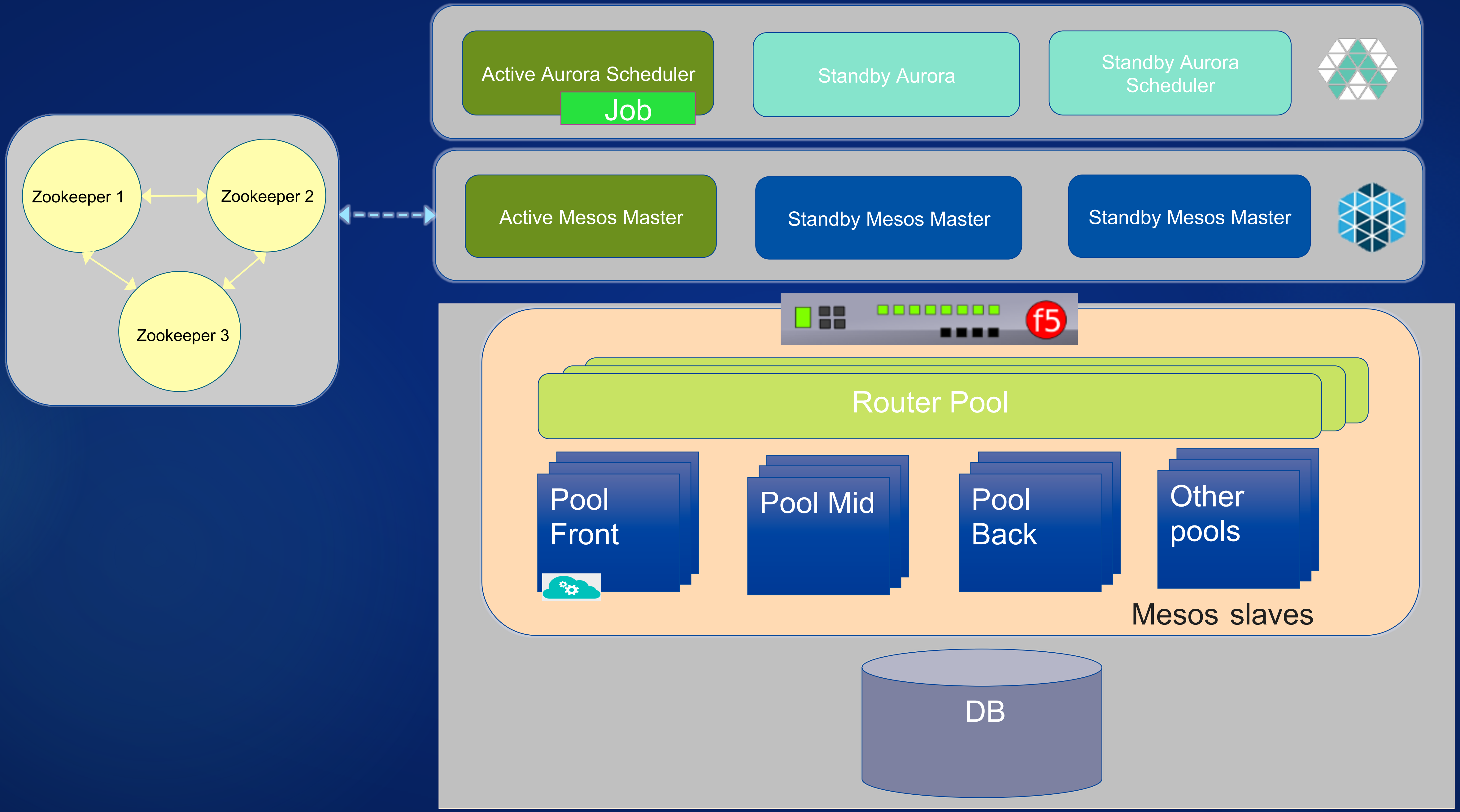
header
  chroot browse
  hostname stage2cs2538
  launch time 08/05 21:42:35
  task config view

task status	time
ACTIVE	08/05 21:42:35

cpu	ram	disk
0.184	10332.3MB	3.5GB

process			time		used		logs	
name	run	status	started	finished	cpu	ram	stdout	stderr
setup_mp	0	SUCCESS	08/05 21:42:35	08/05 21:42:40			stdout	stderr
setup	0	SUCCESS	08/05 21:42:35	08/05 21:42:59			stdout	stderr
setup_columbus	0	SUCCESS	08/05 21:42:35	08/05 21:42:35			stdout	stderr
start_mp	0	RUNNING	08/05 21:42:42		0.337	9705MB	stdout	stderr
install	0	SUCCESS	08/05 21:42:59	08/05 21:56:26			stdout	stderr
start	0	SUCCESS	08/05 21:56:26	08/05 21:58:07			stdout	stderr
monitor	0	RUNNING	08/05 21:58:08		0.000	36MB	stdout	stderr
start_columbus	4	RUNNING	08/07 01:30:07		0.000	590MB	stdout	stderr

Dynamic Service Discovery & Registration



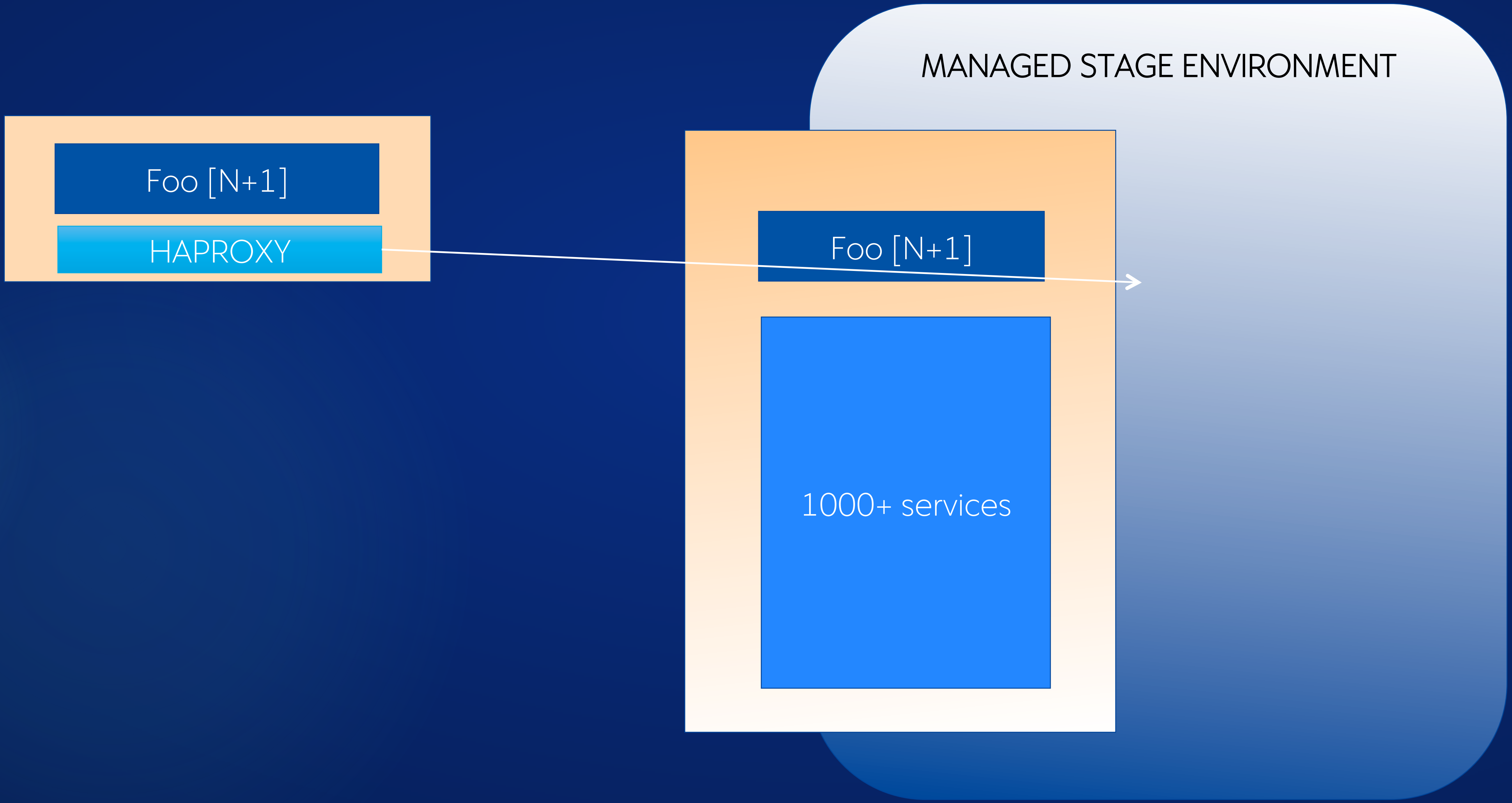
Code Refresh Cycle

- In-place and Full deploy

- 5000 Total number of packages gets deployed
  - Full Deploy takes < 1 Hr
  - Incremental deploy < 20 Mins

- Code refresh frequency :
  - Daily code refresh incrementally
  - Biweekly full code refresh

Dependent Stage

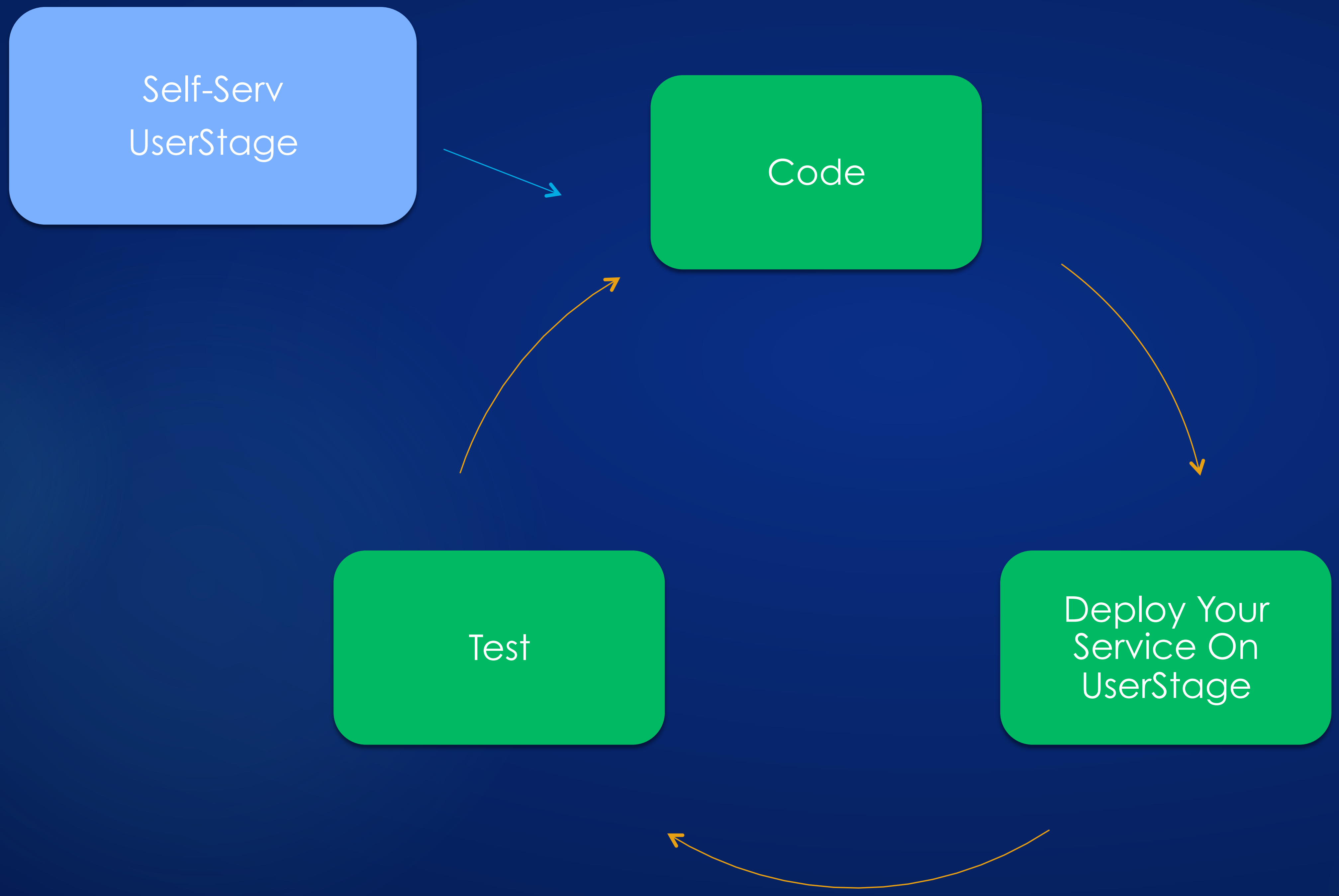


Advantage

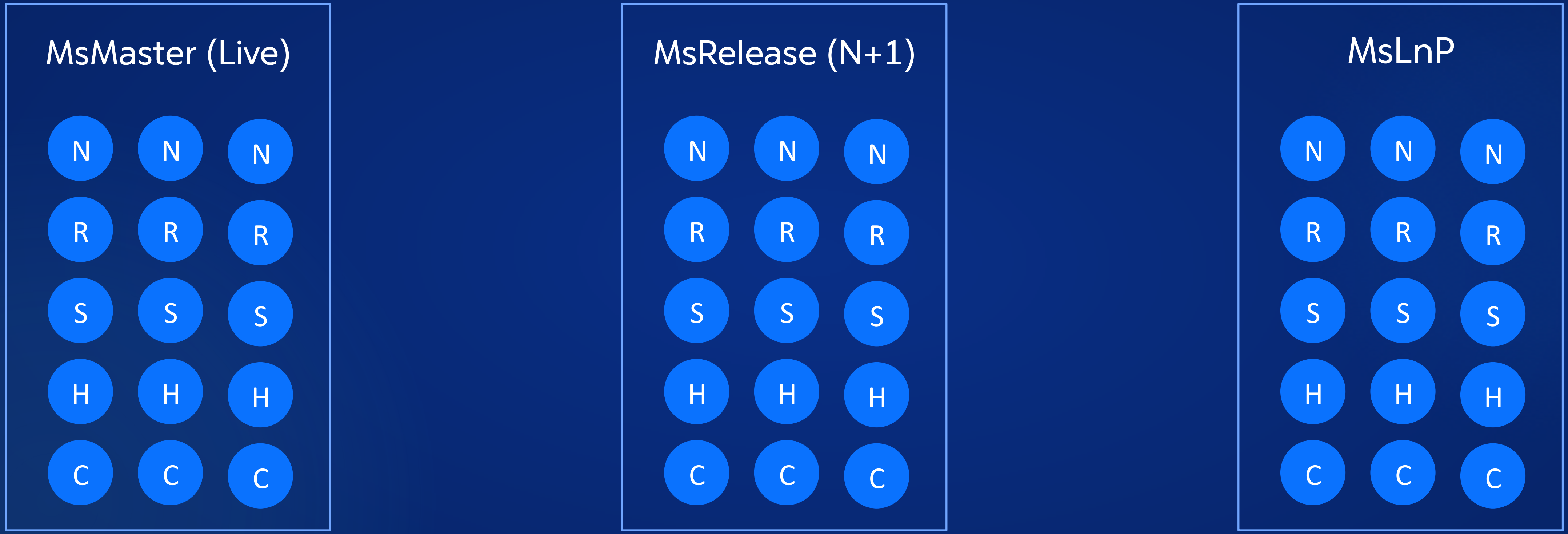
- Developer
  - Stage setup is reduced by 90%
  - Increased productivity by 30 - 40%
  - Abstracting all transitive dependencies

- Infrastructure
  - Optimized resource utilization
  - Reduced hardware cost in data center
  - Less network traffic for deploy

Test Cycle at PayPal After Managed Stage



Managed Stage Versions

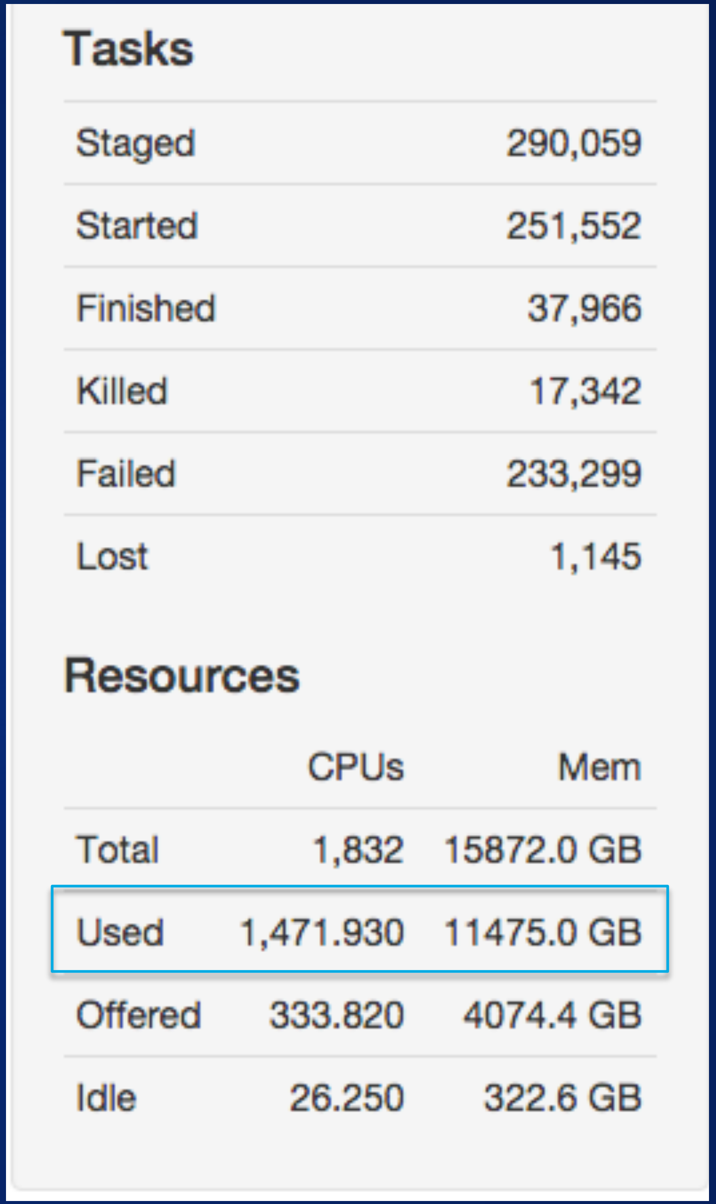


Why Mesos ?

- CI on Mesos was a success at PayPal
- Setup cost and time is really low
- Mesos & Aurora : an out-of-the-box solution
- Docker integration in future

Cost Analysis

- No. of VM : ~ 5000
- Average Stage size : 16 CPU, 64 GB RAM, 256 GB HDD
- Total CPU : 80,000
- Total CPU used in Managed Stage : 1500/env
- Reclaimed CPU : 40,000 (just by 50% reduction)



Tasks

Staged	290,059
Started	251,552
Finished	37,966
Killed	17,342
Failed	233,299
Lost	1,145

Resources

	CPUs	Mem
Total	1,832	15872.0 GB
Used	1,471.930	11475.0 GB
Offered	333.820	4074.4 GB
Idle	26.250	322.6 GB

Roadmap

- Using network isolation which was introduced in Mesos 0.22.0
- Elastic scaling bases on usage patterns
- Merging clusters with CI
- Exploring Docker for containerizing pools

Vision

With a click of a button, you can create your own environment with components of specific versions.

Q&A

Thanks