‘Microservice’ Application Delivery – pipeline
(essential work flow steps)

1 Write application code

2 Test

3 Package:
- src
- config
- security
- compliance
To create a production worthy image
-
And test in Development

4 Provisioning:
Day 1
and Day 2+
-
Infrastructure

5 Deployment:
Scheduler or other Orchestration tool

6 Monitor:
- Logging
- Telemetry
- Tracing
- Health Checking & Alerting
-
We want to be notified if something goes wrong:
Such as Service Outage, Customers can’t reach the site
- telling us that something bad is happening and it is impacting user facing services
- so we really want to be ** ALTERED **

7 **Security** should be applied at each step of the pipeline and not left until after monitoring

Additional steps

Test
- static code analysis
- unit test
- and many other

Package management

Into an:
AMI,
VM or
Docker
container

Test
- integration
- load / stress testing

These two may be one and the same

Database of all inventory, maybe in Source Control

On a large enough scale any production system is a black box and we really can’t possibly understand how the whole thing works.
So what is most important is do we have enough observability, probes into the system to understand what has gone wrong when we get an ALERT.
When we get an ALERT we want to make sure we have enough observability data to fix the problem and get it back online.

Hashicorp tools that can be used at various steps in pipeline to achieve “**Infrastructure as Code**”

These tools have their files in source control and we can achieve automation for every step.

Vagrant : 1, 2
(for VM or Docker to make development look like production via a Vagrant file)

Packer : 3
(to consume all of the inputs and specify via a Packer file what we want as our outputs image)

Terraform : 4
(config file captures what the infrastructure is to look like, a graph of infrastructure that could include:
VPC to define network structure,
Security groups on top of that,
Virtual machines and front it all with a load balancer – just for this ONE application)
You may need to add DNS record in front of the Load Balancer, and then you may need to deploy a content delivery network in front of that. (**Cloudflare, FASTLY**)
Day 1, terraform applies the config’d plan.
Day 2+, terraform applies the delta from new config file’s.

Nomad : 5
Application scheduler (App’ in VM, on top of OS).
App developers care about App’ and Operators care about OS; Nomad sits in between these two.
Job file says what App’ to run, what its version is and how many instances to run.
Nomad looks for resource to go run the required instances as the number required is changed in config file.
Nomad has a variety of roll-out strategies : blue-green, canary, time-staggering (managed through job config file), etc.
It can run up a new fleet with newer OS and tell nomad to migrate from old fleet to new fleet.
(For better resource utilisation, it is better to deploy the App’ into a container).
Nomad can run more than one App’ on a machine, that is multiple VM’s and / or multiple Docker Containers.

Consul : 6
A tool for building service oriented or microservice applications.
Service discovery – a registry of what is running where, load balancing across instances and automatically routing around any failures that happen.
Configuration – to put service A into maintenance mode, etc – via a Key, Value store.
Resiliancy – Leader election, automatic health checks, failover.
To deliver a service oriented architecture.

Vault : 7
Secrets that are everywhere. Vault centralises them, encrypts them at rest and in transit.
Provides strong access control, strong auditing and only distribute secrets on an as needed basis.
Rotating and updating credentials. Exposes cryptographic API’s.
Provides ephemeral dynamic secrets (they time out), to not trust application to not leak them.