Think again before migrating to Kubernetes


grofers

You build it, you run it

Then came Kubernetes

State of Kubernetes at Grofers - 2021

- **Production Environment**
  - 75% of targeted workload migrated to Kubernetes
  - Stateful services and 10% of targeted workload will not be migrated
- **Development Environment**
  - Development in the cloud
  - Kubernetes as extension of laptop for development
  - On-demand dev environments under **10 minutes**

What it means for developers

- Ansible based tooling for stateful workloads in production
- Kubernetes for everything else in production and non-production

You built it, you run it

- Developers get better abstractions that are provided by platform teams
- Platform teams work towards reducing ops overhead at scale. Things like cost, security, reliability become “lesser” of a concern for developers.
- More ownership

What did it take us to get here?

- Two years
- 9 infrastructure engineers
- Collective X months of developer time
- USD 500K of AWS spend

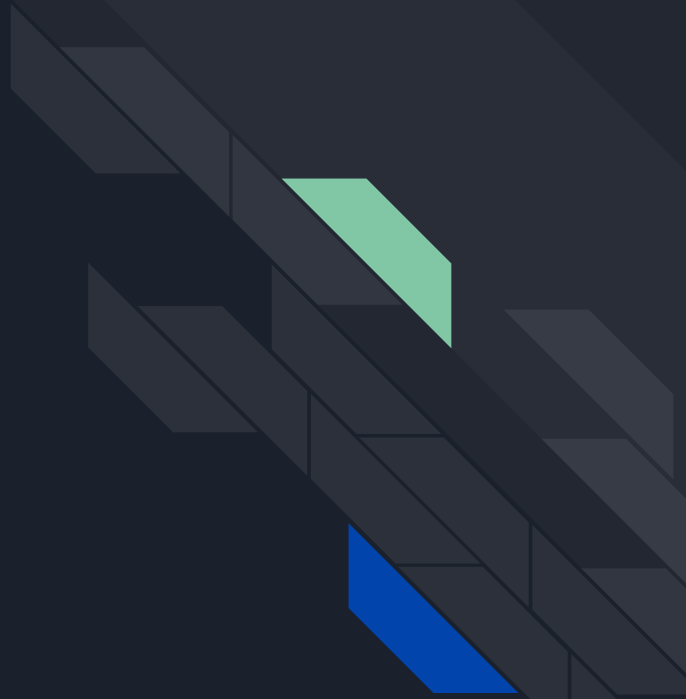
Early 2018

Realized we are dealing with a distributed monolith

Microservices mess - a distributed monolith

<add a visual for microservices mess>

Lesson:
A DevOps strategy requires guardrails for teams to be autonomous while being accountable.



Developers were...

- Unhappy about the drag caused by poor development experience
- Releasing software that was not tested enough
- Fire fighting because of high number of bugs and incidents in production
- Stressed because of sleepless nights and busy weekends

March 2018

Project ShipIt

Project ShipIt

- Automation tests - to run tests end-to-end, we needed to provision all our microservices together for every change.
- Iteratively fix our architecture so that:
  - Teams are truly independent with their microservices.
  - we don’t have to run expensive end-to-end tests.

Journey of Project ShipIt

FEB 2018
Started with Docker and docker-compose

MAR 2018
Ran first set of end-to-end tests on a fully orchestrated container based environment

JUN 2018
Still optimizing docker-compose and our orchestration tooling

AUG 2018
Frustrated with dev/prod disparity.

SEP 2018
Kubernetes decided as the future.

JAN 2019
Kubernetes in production with a few services to prove scale

MAR 2019
We were running tests

Project ShipIt

- Release software faster independently instead of orchestrating deployments
- Release dependencies between teams and fix boundaries of microservices
- Make sure that we are not orchestrating deployments between microservices
- Embrace the value that microservices are supposed to bring - enable teams to be able to develop, test, release and operate services independently.

What are we going to do about it?

- New testing strategy for end-to-end tests for every change until…
- ...we get the boundaries right and can test the changes independently instead of testing the entire thing

Project ShipIt - First Attempt

- To run tests end-to-end, we needed to provision all our microservices together for every change. This was a complex undertaking at the infrastructure level.
- **First attempt** - dockerize every application with docker-compose. We were already using docker-compose for local development and hoped that we can run everything on a large EC2 instance together.
- Ran this for a while to realize that it was too slow and too hard to speed it up enough to reliably run our tests off them.
    - It was brittle
    - It was unmaintainable because of dev/prod dis-parity. My god we had so many production issues because of the drift between the docker-compose and Ansible setup.
- At the end, we spent a couple of quarters chasing an incorrect solution for a complex problem.
- Dev/Prod disparity

Project ShipIt - Second Attempt

- Attack the problem of dev/prod disparity.
- Use tools that were built for complex orchestration instead of having our own.
- Provision test environment using Kubernetes and make sure that we use Kubernetes in production as well to avoid dev/prod disparity.
- This was our reason to migrate to Kubernetes - to solve our ability to test microservices, have a better CI practice and fix our microservices boundaries in the long run.

Find your right reason to migrate to Kubernetes

What are your reasons?

- More agility - consistent developer and DevOps experience, internal DevOps platforms, CI/CD, abstractions for declarative infrastructure management
- Better reliability - better scalability, features for resilience, primitives for streamlined operations.
- Cost - reduced hosting cost and support cost, but most likely a high migration cost.
- Portability - deploy across varied environments (multi cloud, hybrid, etc.)

Is Kubernetes the only way to achieve all of this?

Building more incrementally vs Rebuilding everything

Was migrating to Kubernetes worth the time and money we spent?

~~Was migrating to Kubernetes worth the time and money we spent?~~

Was the process of adopting Kubernetes right for us?

Alternate way

Short Term Track

- Keep using EC2 VMs, Ansible and Consul
- Strengthen service discovery with Consul
- Speed up Ansible further
- Spin up new EC2 VMs

Long Term Track

- Build a solid Kubernetes platform for organic adoption
- Migrate complex infrastructure first to streamline operation
- It’s fine to take more time

Do you really need Kubernetes and if yes, do you need it now?

Embracing Kubernetes

Understand how Kubernetes is really different. Its a building block for platform, not the platform itself.

Costs are going to look very different, so cost control will have to change as well

Kubernetes is a new paradigm. Lift-and-shift is not a good idea.

There is a ***steep learning curve*** for your development and operations teams.

Out-of-the-box Kubernetes is almost never enough for anyone

It is not a PaaS solution but
a building block to build your own PaaS solution

Your Kubernetes platform will shape according to your context

It is a reflection of your:

- Product & Business Context
- Engineering Team
- Existing applications and their infrastructure
- Ability to spend money to migrate to a new way of working

PaaS with Kubernetes

- Configuration & Secret Management
- Development Experience
- Packaging, Deployment
- Continuous Integration
- Continuous Delivery (+ GitOps)
- Metrics
- Logs
- Distributed Tracing
- Governance
- Cost

PaaS with Kubernetes

Configuration & Secret Management	Consul, Vault, consul-template
Development Experience	Skaffold, Telepresence.io
Packaging, Deployment	Skaffold, Kustomize, Helm
Continuous Integration	Tekton
Continuous Delivery (+ GitOps)	Tekton
Metrics	Prometheus, Grafana, NewRelic
Logs	Loki, Grafana
Distributed Tracing	*Not solved yet*
Governance	OPA and Kyverno
Cost	Self-managed spot instances

You are limited by your
legacy applications.

What about adoption and migration?

Not enough incentive
to migrate and take the risk

Better platform is the best incentive to migrate

Lack of proactive support made didn’t help with

Running parallel stacks is hard

Knowledge and operational support both are important for a successful migration and transformation.

Operating a Kubernetes cluster is hard.

CRDs, Operators, Controllers, Mutating Webhooks

Reminder - do you need to do this now?

So where are we going now?

The platform must evolve

- Treat it like a product that solves problems for its users i.e. developers
- Understand that developers will have new unsolved problems that will have to be solved in the future
- Existing solutions will improve in your context and in the community
- There is really no one good answer

A multi-year roadmap

- Infrastructure and application architected together
- Golden path
- Resilient and secure by design

Thank You!

Questions?

Vaidik Kapoor

@vaidikkapoor
linkedin.com/in/vaidik
vaidik.in