In-Cluster Development Environments using Tilt

Nicholas Capo

Senior Infrastructure Engineer

Axios, Inc

Overview

1. Run It Locally
2. Docker Compose
3. Local `tilt`
4. Remote `tilt`

Caveat

This might require a Platform/Infrastructure Team

Goals

- **Ease of Use**: Can we make this a single command?
- **Supportable**: What happens when it breaks?
- **Run Everything**: Can I run all the services at the same time?
- **Production Like**: Minimize differences to reduce surprise as we approach `prod`

  `dev` -> `preview` -> `test` -> `prod`

What is Production Like (today)?

- Kubernetes
- Docker images
- Environment variables
- Helm chart that generates all manifests
- Service mesh (ish)
- Mix of Golang, Python, NodeJS

Run It Locally

- We all know how to do this (mostly)

```
go run ./main.go
```

- Live reload with `air`

```
go install github.com/air-verse/air@latest
air -c air.toml
```

- Needs some docs for each service (Python, NodeJS)
- What Environment Variables should be set?
- Docs need to be maintained

> How do I push changes to someone's mental model?

Goals: Run It Locally

Ease of Use	Supportable	Run Everything	Production Like
Experienced Developer	Mostly	Possibly?	Nope!

Docker Compose

```
# docker-compose.yaml
version: "2"
services:
  web:
    env_file:
      - dev.env
    depends_on:
      - db
      - es
  db:
    image: postgres
  es:
    image: elasticsearch
```

Goals: Docker Compose

Ease of Use	Supportable	Run Everything	Production Like
Pretty good	Git repo, reproducible	Limited Resources	Not quite

Local Tilt

Tilt: https://tilt.dev/

1. `docker build`
2. `docker push`
3. Apply Kubernetes Manifests
4. Live Update of source after running

Runtime can be `docker-compose` / `k3d` / `minikube` / EKS / AKS / etc

Setup script

Needs a little bit of `setup`

- General setup
  - VPN
  - AWS auth
- Start Cluster
- Start Tilt

Goals: Local Tilt

Ease of Use	Supportable	Run Everything	Production Like
Single Command!	Git repo, reproducible	Limited Resources	Mostly!

> Limited Resources is more of a problem than we thought!

The Disadvantages of `localhost`

- `http://localhost:80`
- Subdomains
- Browser features that require TLS
- Self-signed certs and browser trust
- Endpoint Firewalls
- Browser based attacks against `localhost` services

Remote `tilt`

> We already have a shared AWS EKS cluster for staging,
> let's re-use that for dev environments

Some (additional) setup required:

- Namespace-per-developer: `dev-ncapo`
- RBAC for full control of the namespace

Advantages

- Unified Observability
- Unified Ingress
- Visible to other developers
- Visible to the Infrastructure Team
- (Mostly) unlimited resources

Goals: Remote `tilt`

Ease of Use	Supportable	Run Everything	Production Like
Single Command	Yes	Completely	Closest we could get

Refinements

- Dependency Caching
- Ongoing Maintenance

Dependency Caching

> `node_modules` is a huge black hole, and now we need to `docker push` it across the continent every time we build.
> But actually it's almost the same for GOMODCACHE

Solution

- Change Deployment to `strategy: Recreate`
- Add a PVC with directories mounted on
  - `node_modules` and `~/.npm`
  - `/go/pkg/mod` and `~/.cache/go-build`
- Change entrypoint to compile and then run (`nextjs` / `gunicorn` / `air`)

Ongoing Maintenance

- These `dev` Environments are now a sort of internal production, that needs support and maintenance.
- When the environment goes down people can't do their jobs.

Demo

- `main.go`
- `air.toml`
- `Dockerfile`
- `manifests.yaml`
- `Tiltfile`
- `setup.sh`