



KubeCon



CloudNativeCon

Europe 2023



TiKV



KubeCon



CloudNativeCon

Europe 2023

Tilt Your World! Lessons Learned in Improving Dev Productivity with Tilt

- Yuvraj Kakaraparthi & Sagar Muchhal

About Us



Yuvaraj Kakaraparthi

Sr MTS @VMware

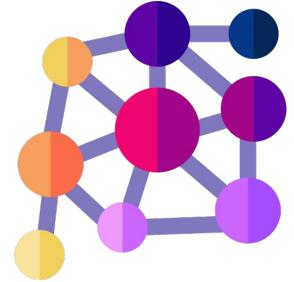


Sagar Muchhal

Staff @VMware

The Problem

Modern applications are becoming increasingly distributed and complex.



*Managing applications and infrastructure is great with Kubernetes but the **development workflow is not.***

The Problem

What makes a good development workflow?

What makes a good development workflow?

 Easy spin up

 Quick feedback loop

 Debuggable

 Visibility

What makes a good development workflow?

 Easy spin up

 Quick feedback loop

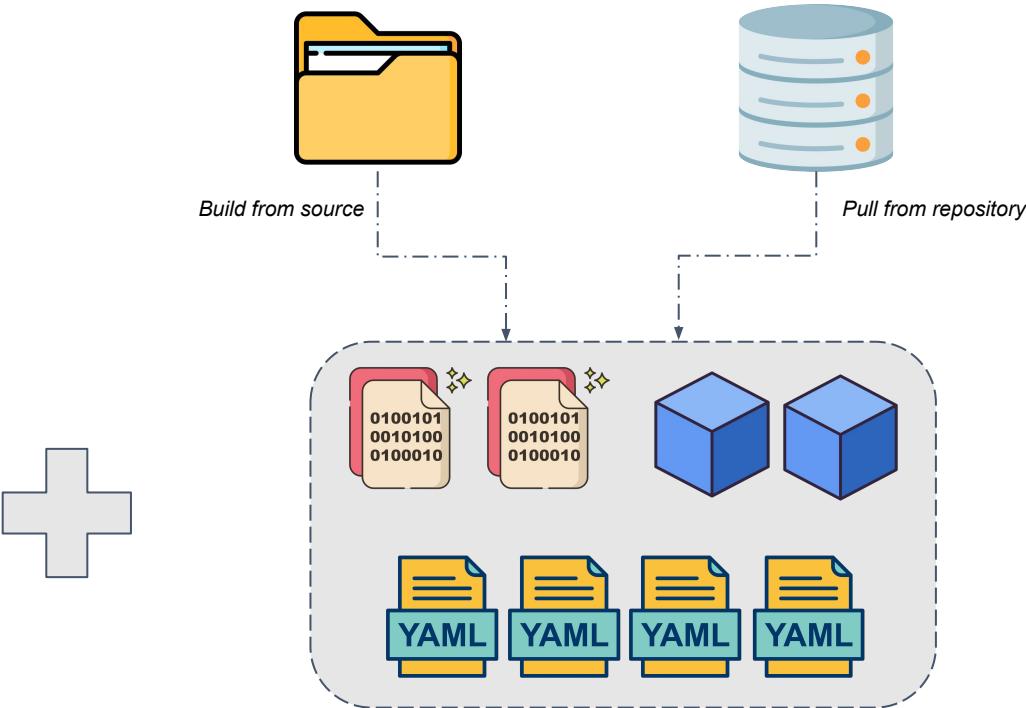
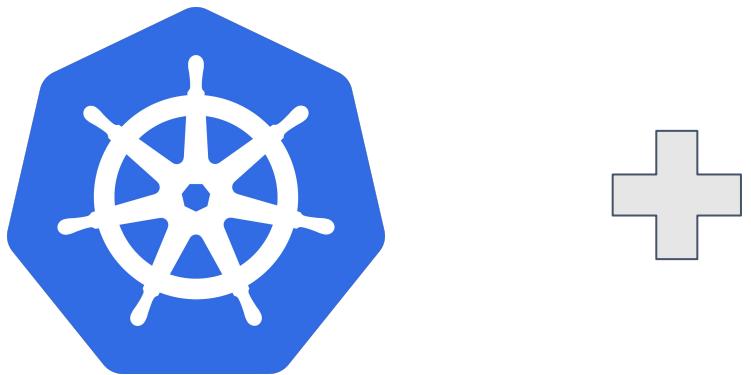
 Debuggable

 Visibility

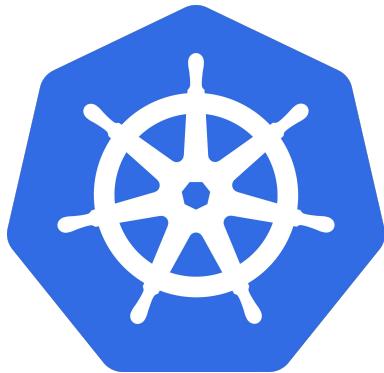
Spin up a development environment

Creating a working application environment from the desired source code.

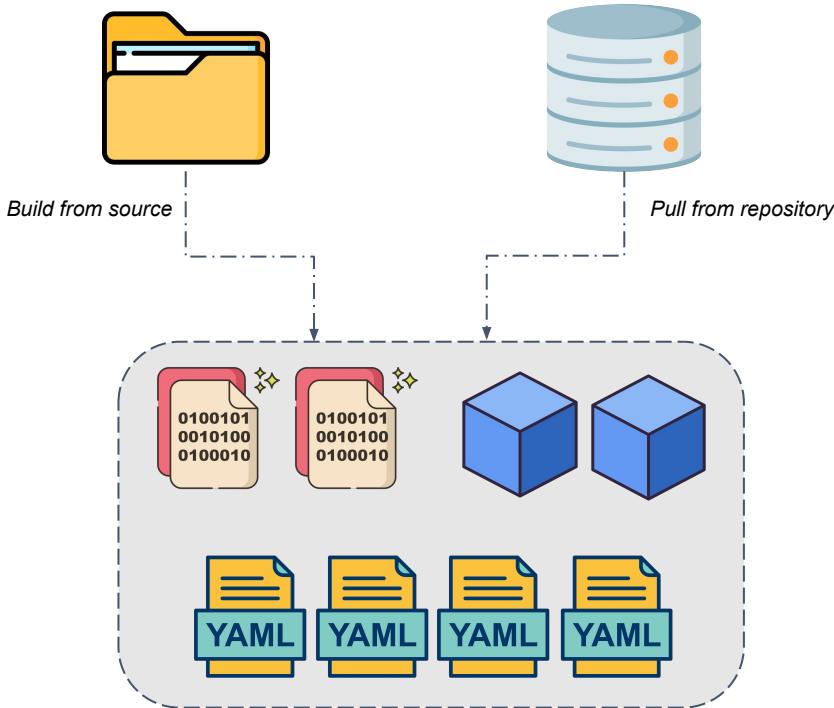
Spin up a development environment



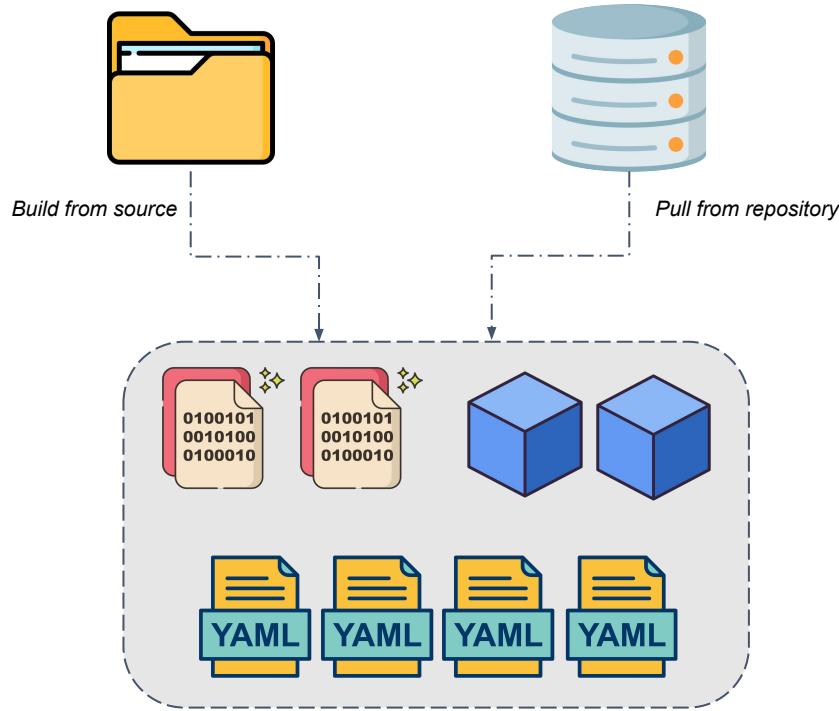
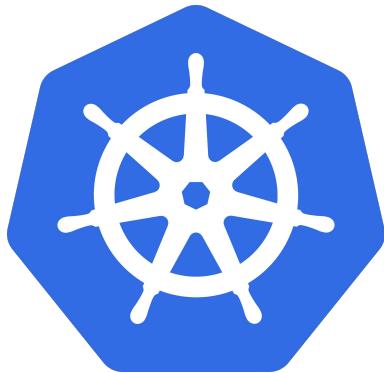
Spin up a development environment



Local Cluster or a
Remote Cluster
depending on the size
of your application.

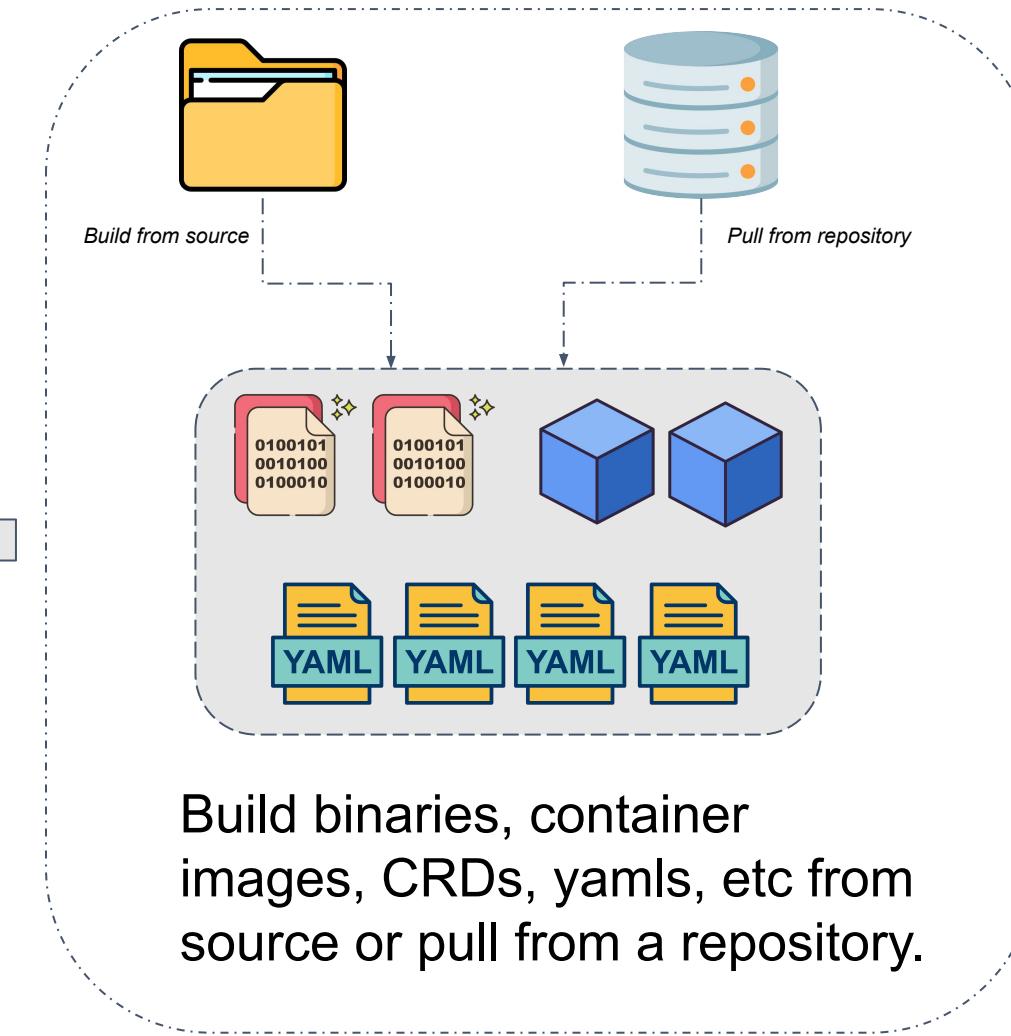
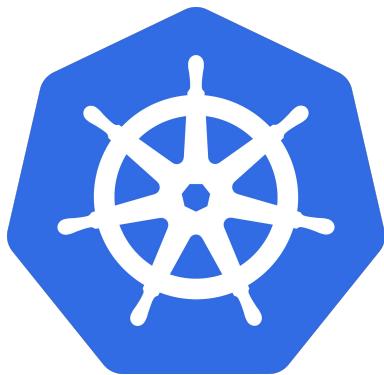


Spin up a development environment



Build binaries, container images, CRDs, yamls, etc from source or pull from a repository.

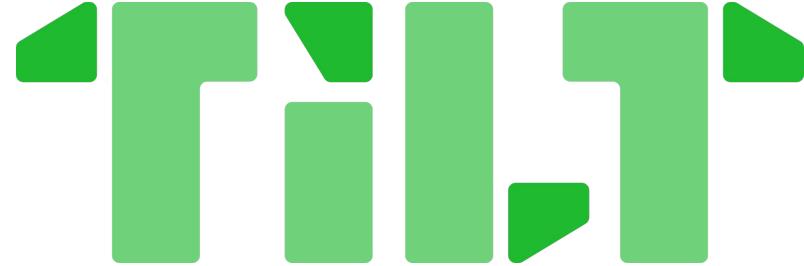
Spin up a development environment



Build binaries, container images, CRDs, yaml's, etc from source or pull from a repository.

*Should be as simple as running a **single command**.*

Spin up a development environment

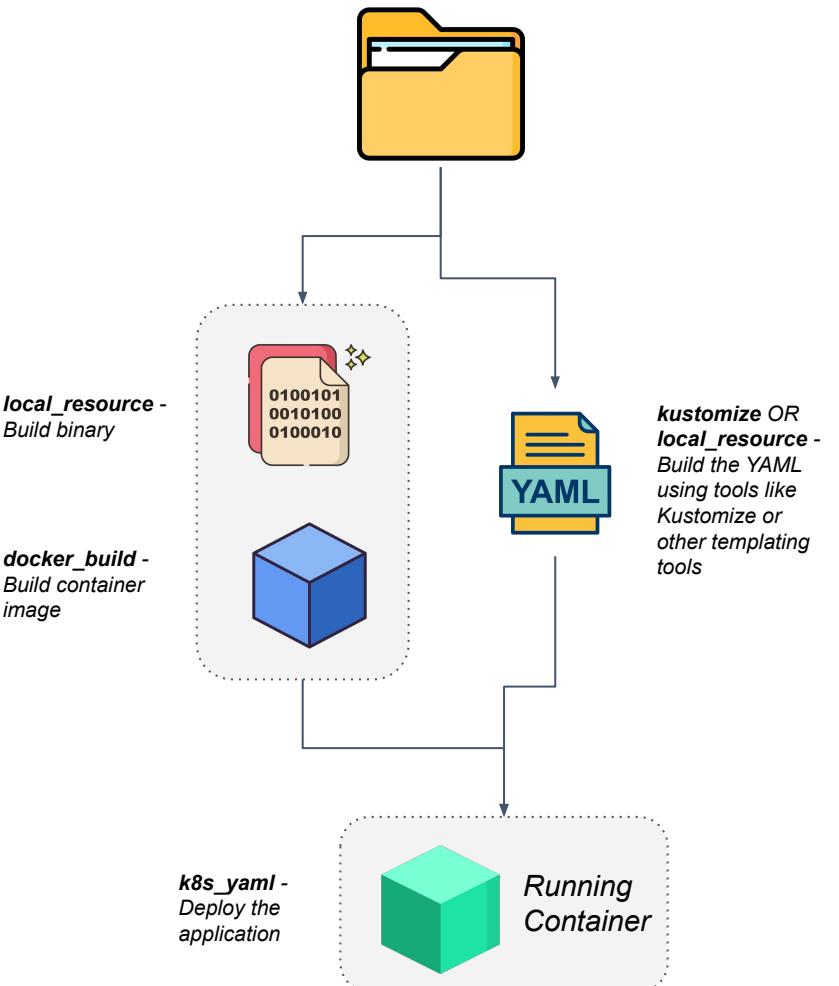


*A toolkit for fixing the pains of developing microservices
... and more.*

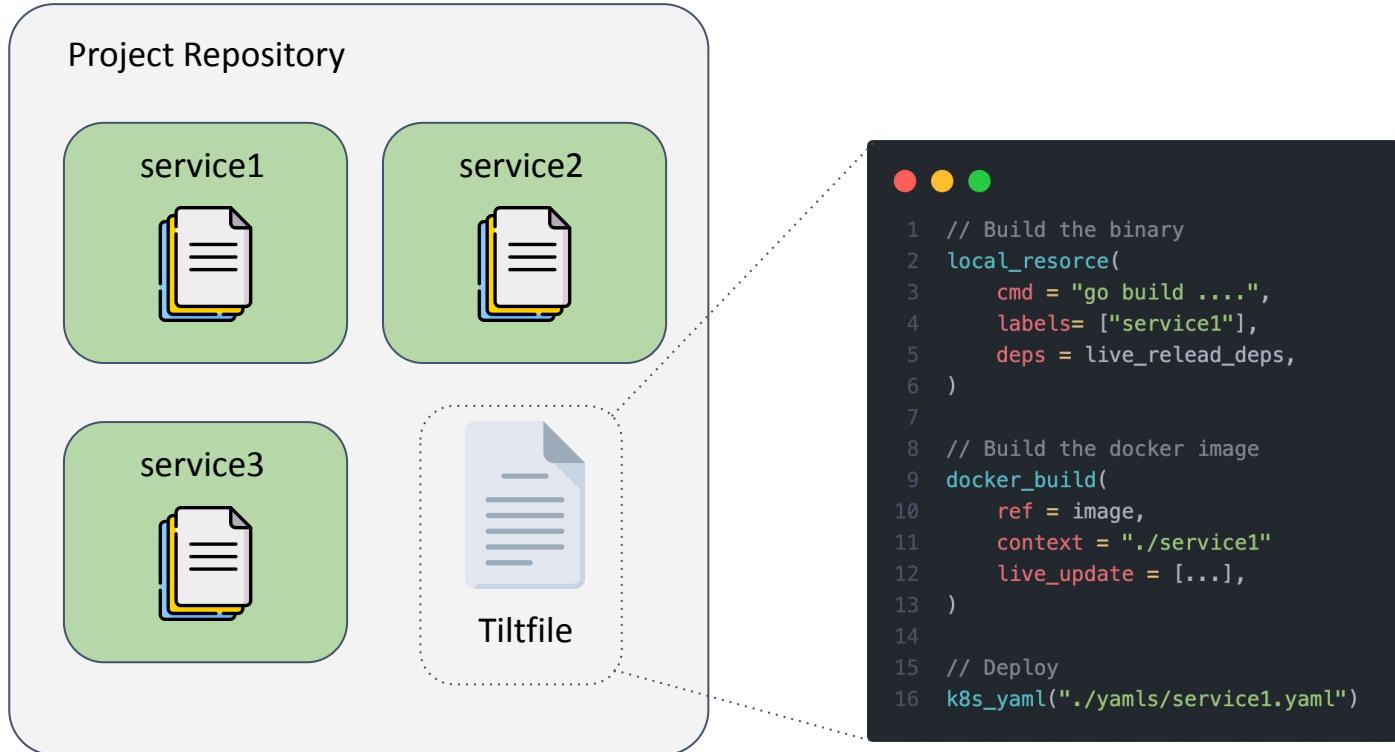
Spin up a development environment



Use a single command `tilt up`
to spin up a development environment.

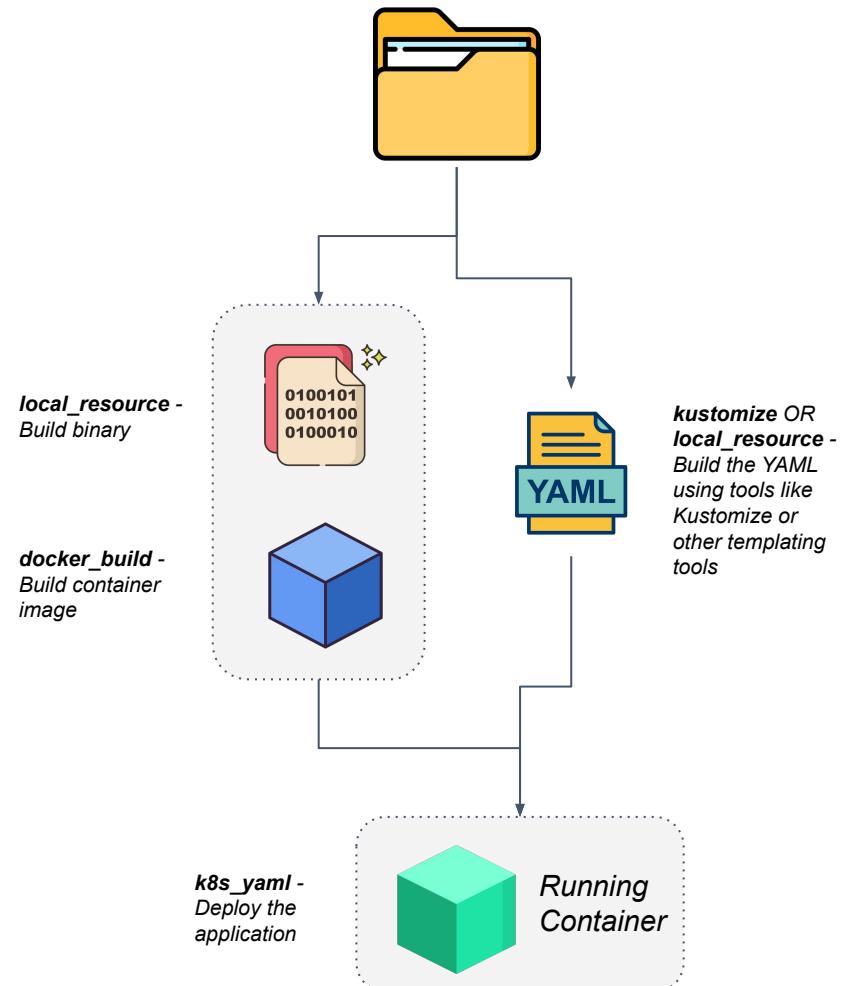


Spin up a development environment



Tiltfile:

- Written in Starlark - a dialect of Python
- Full power of conditions, loops, functions, etc



What makes a good development workflow?

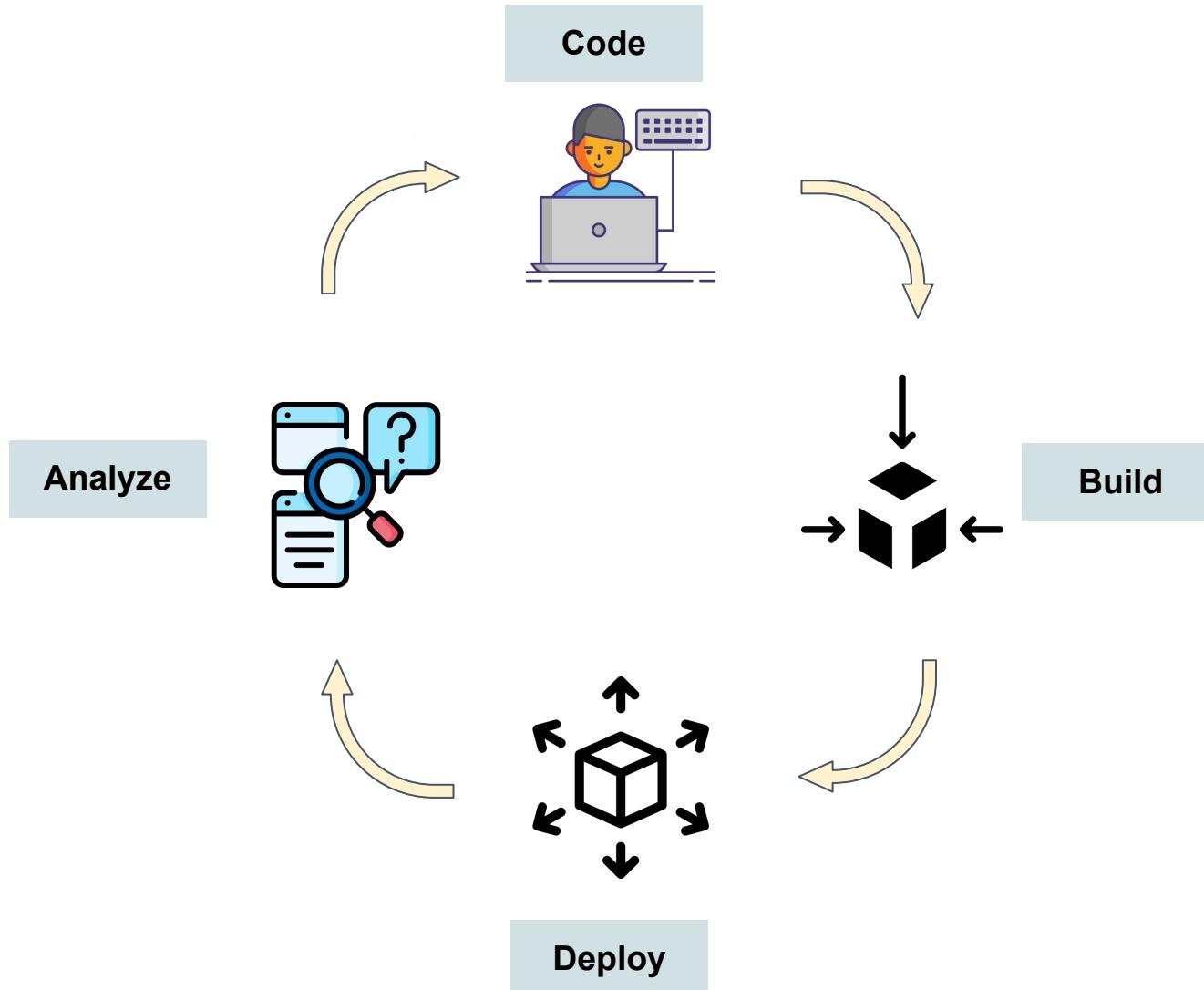
 Easy spin up

 Quick feedback loop

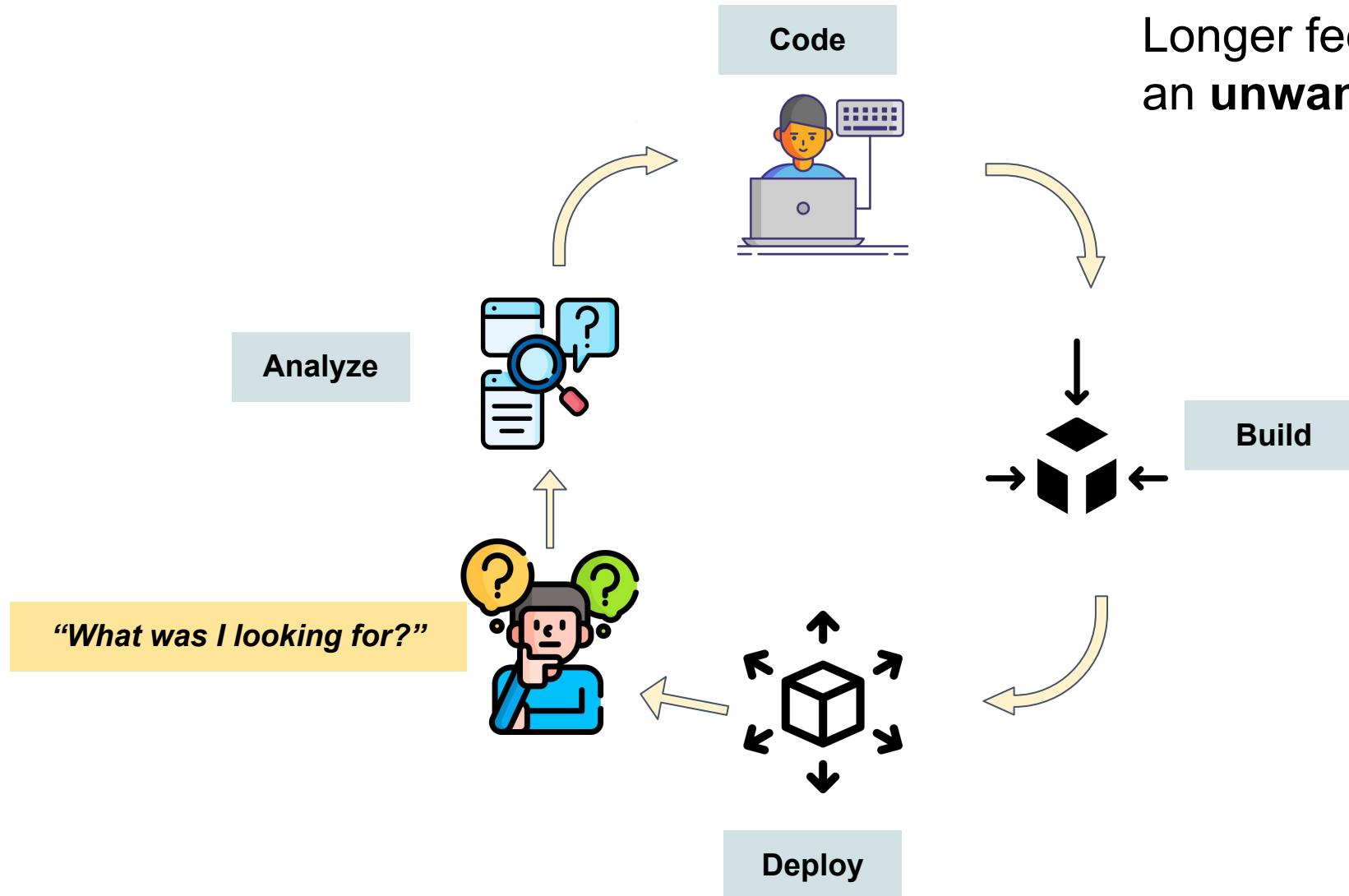
 Debuggable

 Visibility

Feedback loop



Feedback loop



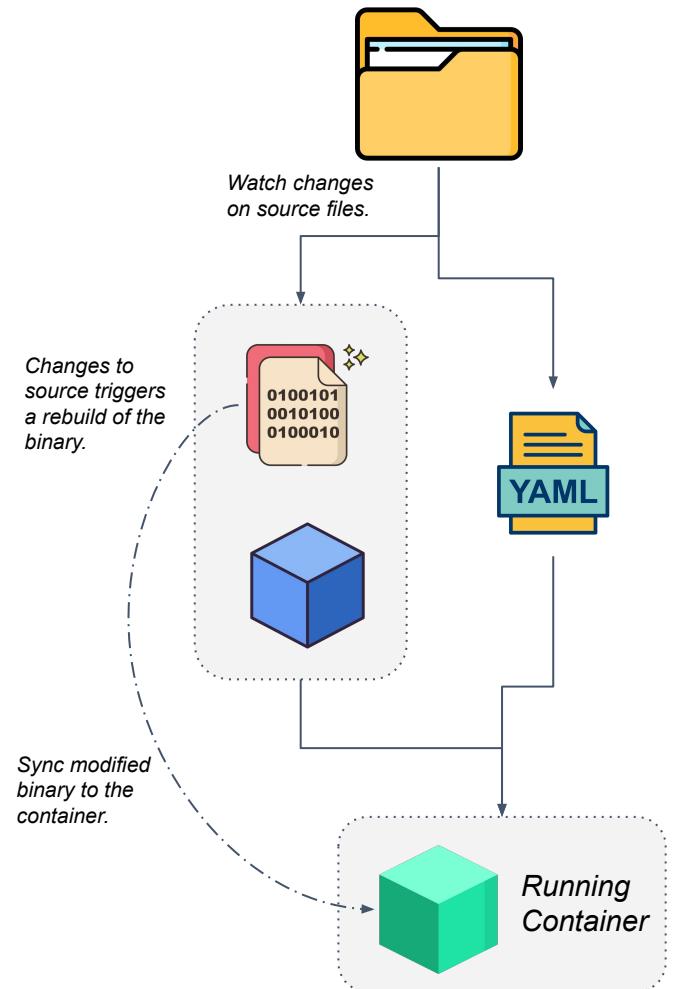
Longer feedback loops add an **unwanted step**.

Feedback loop

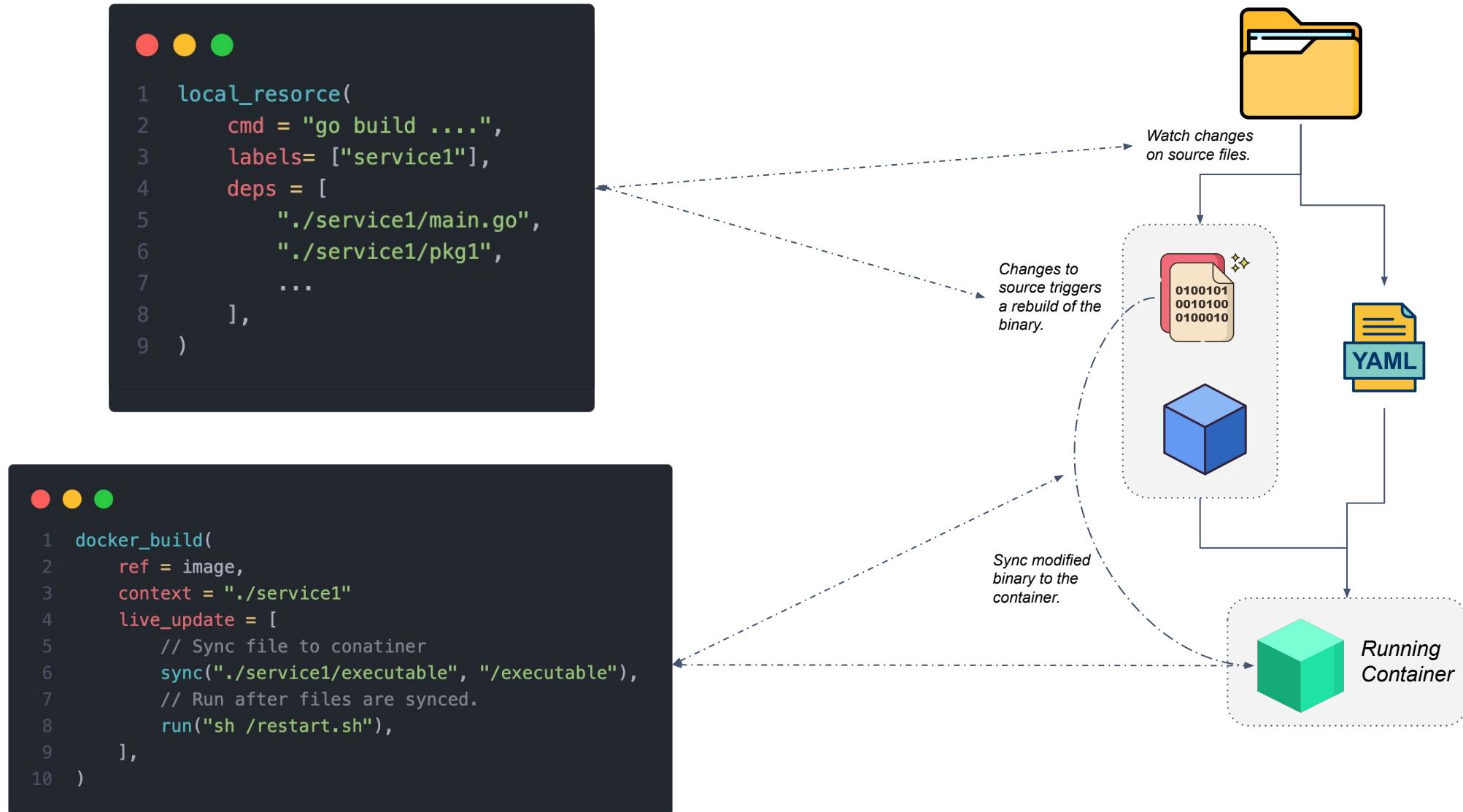


Use **Live Update** to hot-reload the application.

Cut down the feedback loop and see changes as close to real-time as possible.



Feedback loop

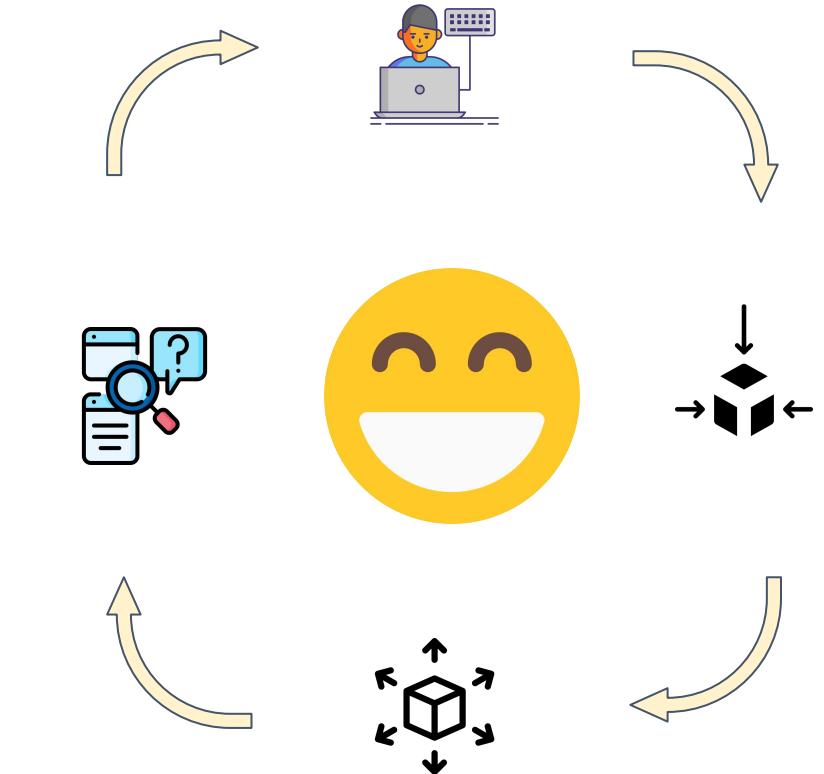


Feedback loop



No more:

- Rebuild container image `docker build`
- Publish container image `docker push`
- Update image tag in YAML `kubectl apply -f ...`
- Wait for pods to rollouts



What makes a good development workflow?

 Easy spin up

 Quick feedback loop

 Debuggable

 Visibility

*The first place most start debugging is by **adding logs***

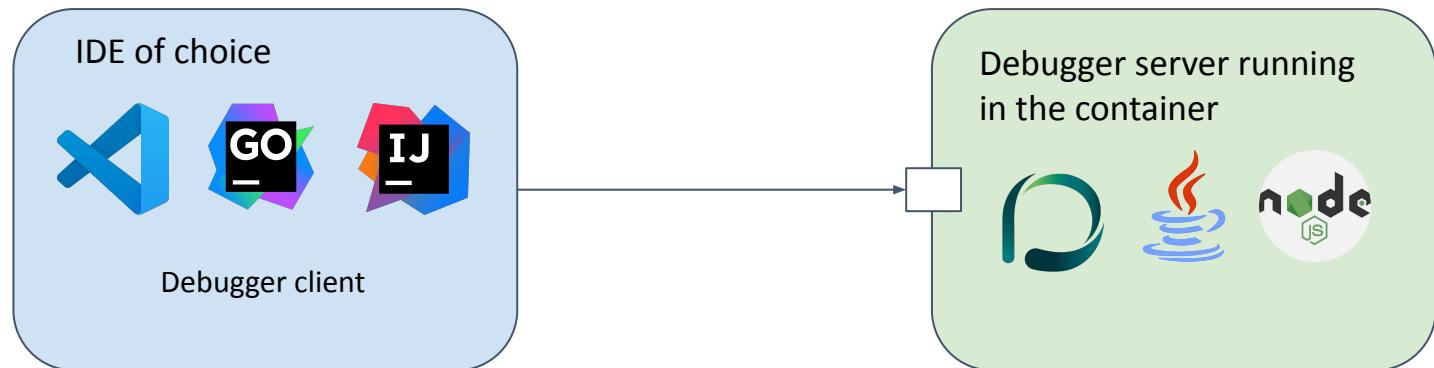
*Quickly becomes **overwhelming** and **hard to debug***

*Using a **debugger** to step through code is the better option*

... but can get tricky when working with containers.

Remote Debugging

- Run a debug server remotely
- Use your local client (IDE) to connect to the remote debug server



Debugging

- Build a debugging compatible binary
- Modify the Deployment
- Port-forward to the debug server port
- Connect to the remote server from the debug client

Debugging

- **Build a debugging compatible binary**
 - Disable compile time optimizations
 - Retain the debugging data (DWARF)
- Modify the Deployment
- Port-forward to the debug server port
- Connect to the remote server from the debug client

Debugging

- Build a debugging compatible binary
- **Modify the Deployment**
- Port-forward to the debug server port
- Connect to the remote server from the debug client

- Change the deployment container's command to start the debugging server

```
● ● ●  
1 apiVersion: apps/v1  
2 kind: Deployment  
3 metadata:  
4   name: service1  
5 spec:  
6   ...  
7     containers:  
8       - name: service1  
9         command:  
10        - sh  
11        - /start.sh  
12        - /dlv  
13        - --accept-multiple  
14        - --api-version=2  
15        - --headless=true  
16        - exec  
17        # Debugger server port  
18        - --listen=:30000  
19        - --  
20        - /executable  
21        args: ...  
22        image: ...
```

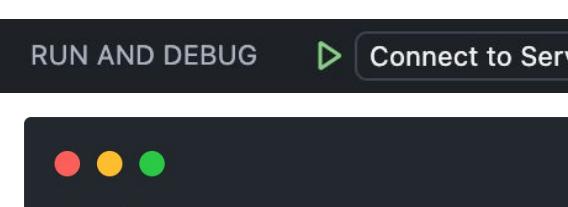
- Disable liveness and readiness probes

Debugging

- Build a debugging compatible binary
 - Setup port forwarding from a free local port to the debug port of the container
- Modify the Deployment
- Port-forward to the debug server port
 - Connect to the remote server from the debug client

```
● ● ●  
1 k8s_resource(  
2   workload = "service1",  
3   labels = ["service1"],  
4   // Port forward to the debug on the container  
5   port_forwards = [  
6     port_forward("30000", "30000"),  
7   ],  
8 )
```

Debugging

- Build a debugging compatible binary
 - Connect to the remote debugger using the debug client (built-into most IDEs)
 - Modify the Deployment
 - Port-forward to the debug server port
 - **Connect to the remote server from the debug client**

```
RUN AND DEBUG > Connect to Server1 ▾ ⚙
```

```
1  {
2      "version": "0.2.0",
3      "configurations": [
4          {
5              "name": "Connect to Server1",
6              "type": "go",
7              "request": "attach",
8              "mode": "remote",
9              // Port setup for port-forwarding
10             "remotePath": "",
11             "port": 30000,
```



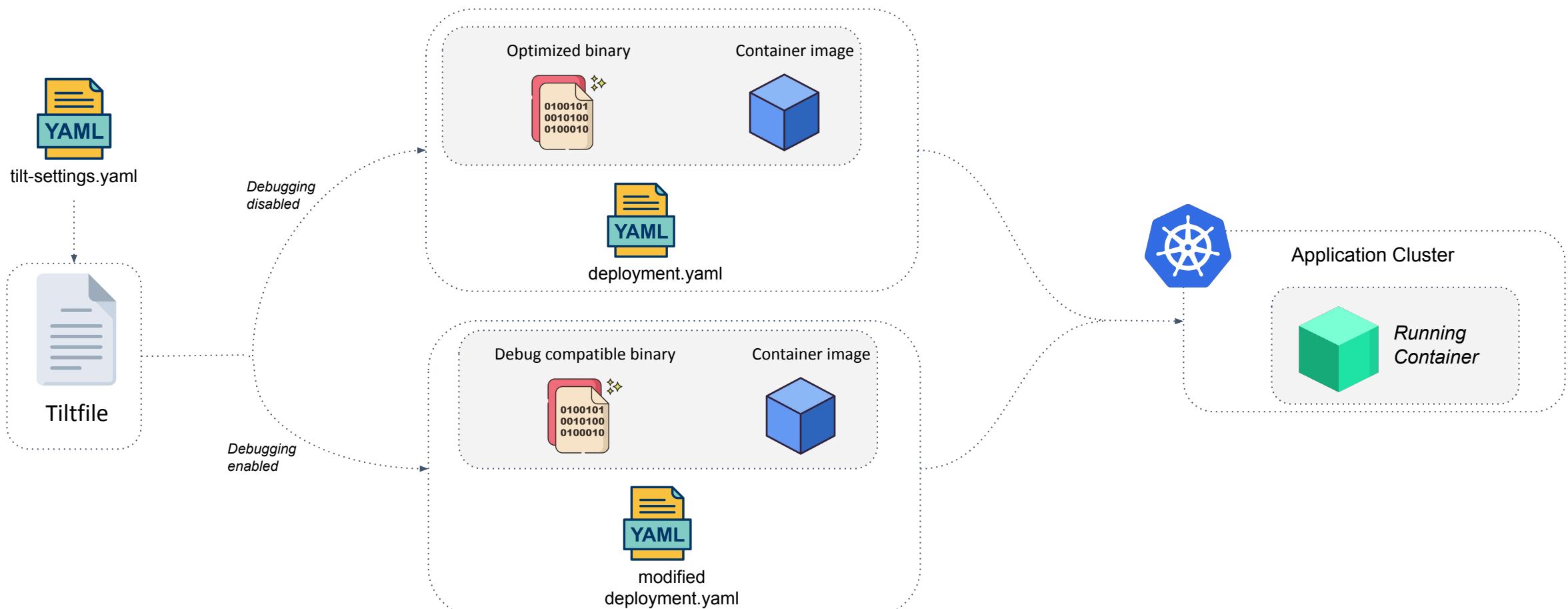
The screenshot shows a software interface with a dark theme. At the top, there's a navigation bar with the text "RUN AND DEBUG" on the left, a green play button icon, the text "Connect to Server1" with a dropdown arrow, a gear icon for settings, and three vertical dots for more options. Below this is a toolbar with three colored circles (red, yellow, green). The main area is a code editor displaying a JSON configuration file:

```
1  {
2      "version": "0.2.0",
3      "configurations": [
4          {
5              "name": "Connect to Server1",
6              "type": "go",
7              "request": "attach",
8              "mode": "remote",
9              // Port setup for port-forwarding
10             "remotePath": "",
11             "port": 30000,
12             "host": "127.0.0.1"
13         }
14     ]
15 }
```

Debugging

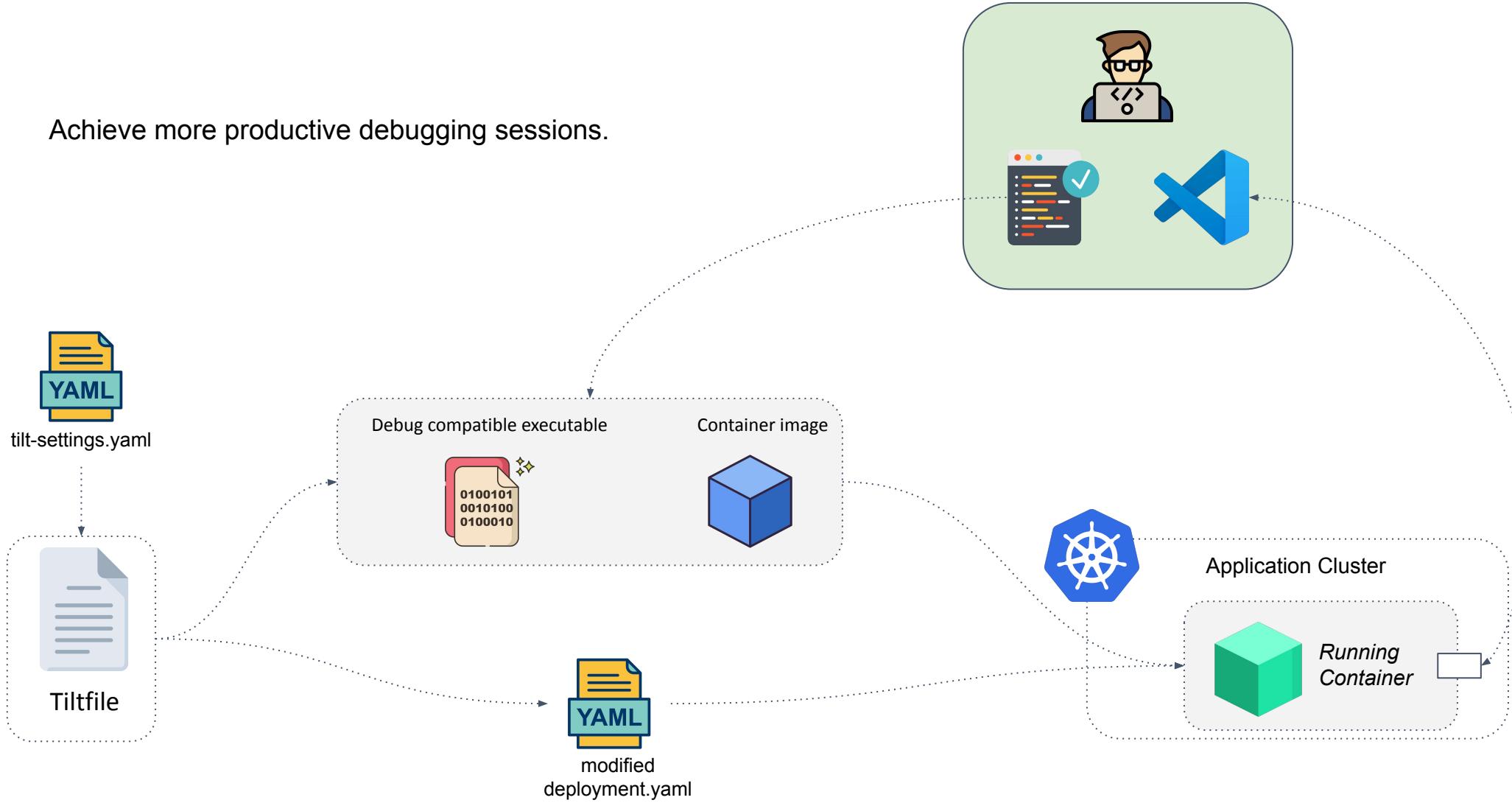
No need to run the debugger all the time. Enable it on demand.

Have a `tilt-settings.yaml` to configure debugging parameters.



Debugging

Achieve more productive debugging sessions.



What makes a good development workflow?

 Easy spin up

 Quick feedback loop

 Debuggable

 Visibility

Visibility



Showcases overall status of the application

A screenshot of the Tilt application interface. At the top, there's a navigation bar with the Tilt logo, a grid icon, a list icon, and a search bar labeled "Filter resources by name". To the right, it shows "RESOURCES ✓ 11 / 11". Below the header, there's a list of resource categories, each with a status indicator (green checkmark) and a count:

- ▶ ALL.binaries ✓ 4 / 4
- ▶ ALL.controllers ✓ 4 / 4
- ▶ CABPK ✓ 2 / 2
- ▶ CAPD ✓ 2 / 2
- ▶ CAPD.clusterclasses
- ▶ CAPD.templates
- ▶ CAPI ✓ 2 / 2
- ▶ KCP ✓ 2 / 2
- ▶ unlabeled ✓ 2 / 2
- ▶ Tiltfile ✓ 1 / 1

Visibility



Showcases overall status of the application

Follow live updates to the components

The screenshot shows the Tilt application interface. At the top, there's a navigation bar with the Tilt logo, resource count (4), and a green success indicator (7/11). Below the bar are filter options for alerts, disabled resources, and log levels (All Levels, Errors). A search bar allows filtering by name or regex. The main area is divided into sections: **All Resources**, **CAPD**, **CAPI**, **TILTFILE**, and **Initial Build**. The **Initial Build** section displays a log of build steps:

```
even non-root users to run the image'

FROM gcr.io/distroless/base:debug as tilt
WORKDIR /
COPY --from=tilt-helper /process.txt .
COPY --from=tilt-helper /start.sh .
COPY --from=tilt-helper /restart.sh .
COPY --from=tilt-helper /go/bin/dlv .
COPY $binary_name .

Building image
[tilt 1/7] FROM
gcr.io/distroless/base:debug@sha256:357bc96a42d8db2e4710d8ae6257da3a66b1243affcd
932438710a53a8d1ac6
[background] read source files 74.41MB [done: 741ms]
[tilt-helper 1/3] FROM docker.io/library/golang:1.19.6
[tilt-helper 2/3] RUN go install github.com/go-delve/delve/cmd/dlv@latest
[cached]
[tilt-helper 3/3] RUN wget --output-document /restart.sh --quiet
https://raw.githubusercontent.com/tilt-dev/re-run-process-
wrapper/master/restart.sh && wget --output-document /start.sh --quiet
https://raw.githubusercontent.com/tilt-dev/re-run-process-wrapper/master/start.sh
&& chmod +x /start.sh && chmod +x /restart.sh && chmod +x /go/bin/dlv &&
touch /process.txt && chmod 0777 /process.txt '# pre-create PID file to allow
even non-root users to run the image' [cached]
[tilt 2/7] COPY --from=tilt-helper /process.txt . [cached]
[tilt 3/7] COPY --from=tilt-helper /start.sh . [cached]
[tilt 4/7] COPY --from=tilt-helper /restart.sh . [cached]
[tilt 5/7] COPY --from=tilt-helper /go/bin/dlv . [cached]
[tilt 6/7] COPY . [cached]
exporting to image [done: 15ms]

STEP 2/3 - Pushing localhost:5000/gcr.io_k8s-staging-cluster-api_cluster-api-
controller:tilt-87ab5b7e6a4d954
Pushing with Docker client
Authenticating to image repo: localhost:5000

admin@Default: ~ [WithDefault: user@localhost:5000]
```

Visibility



Showcases overall status of the application

Follow live updates to the components

Provides detailed information for each component

The screenshot shows the Tilt web interface with the following details:

- Top Bar:** RESOURCES ✓ 11 / 11, v0.31.1, Clear Logs.
- Search Bar:** Filter resources by name.
- Alerts on top:** Alerts on top, Show disabled resources.
- Resource Groups:** All Resources, ALL.binaries (✓ 4 / 4), CABPK (✓ 2 / 2), CAPD (✓ 2 / 2), CAPD.clusterclasses, CAPD.templates, CAPI (✓ 2 / 2).
- Logs:** Web Trigger logs (e.g., controller-runtime/webhook: Conversion webhook enabled, Registering a mutating webhook, Registering a validating webhook).
- Recent Runs:** capi_controller (14m ago, Completed in 4.5s), capi_binary (3d ago, Completed in 1.3s), KCP (✓ 2 / 2), unlabeled (✓ 2 / 2).
- TILTFILE:** (Tiltfile) (2m ago, Completed in 15s).
- Bottom Log Panel:** Detailed log entries for each run, including JSON messages from controller-runtime/webhook.

Visibility

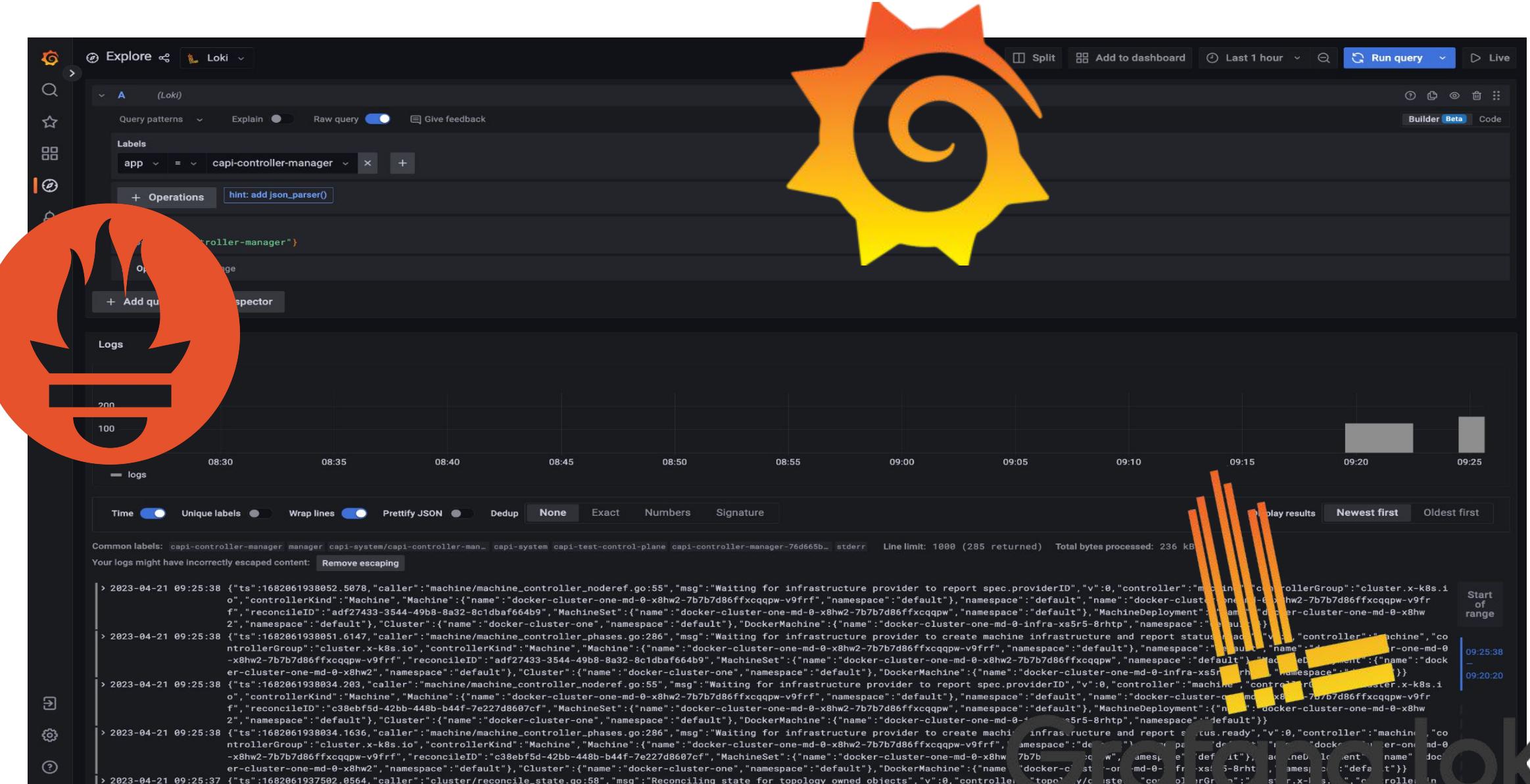


KubeCon

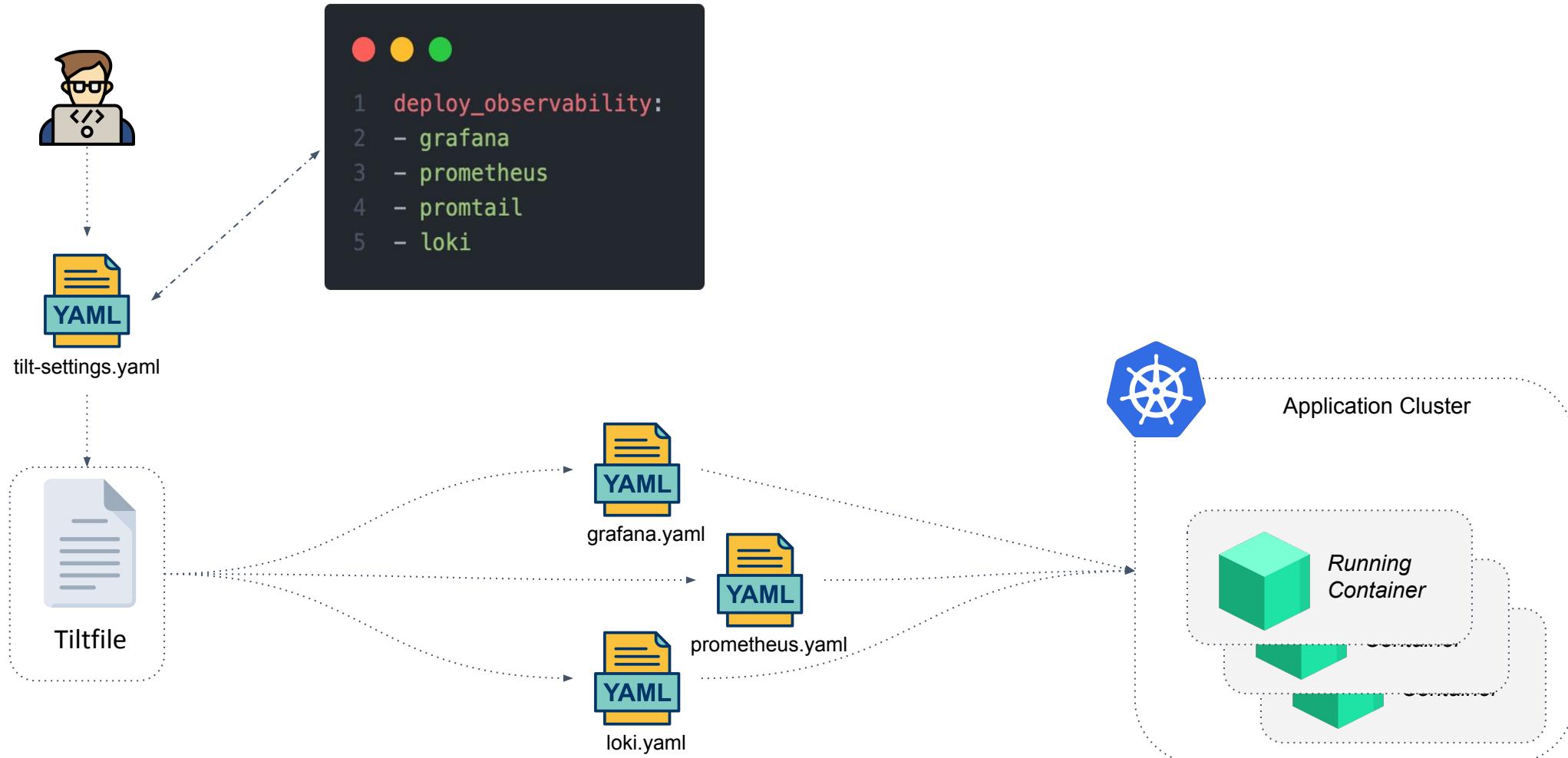


CloudNativeCon

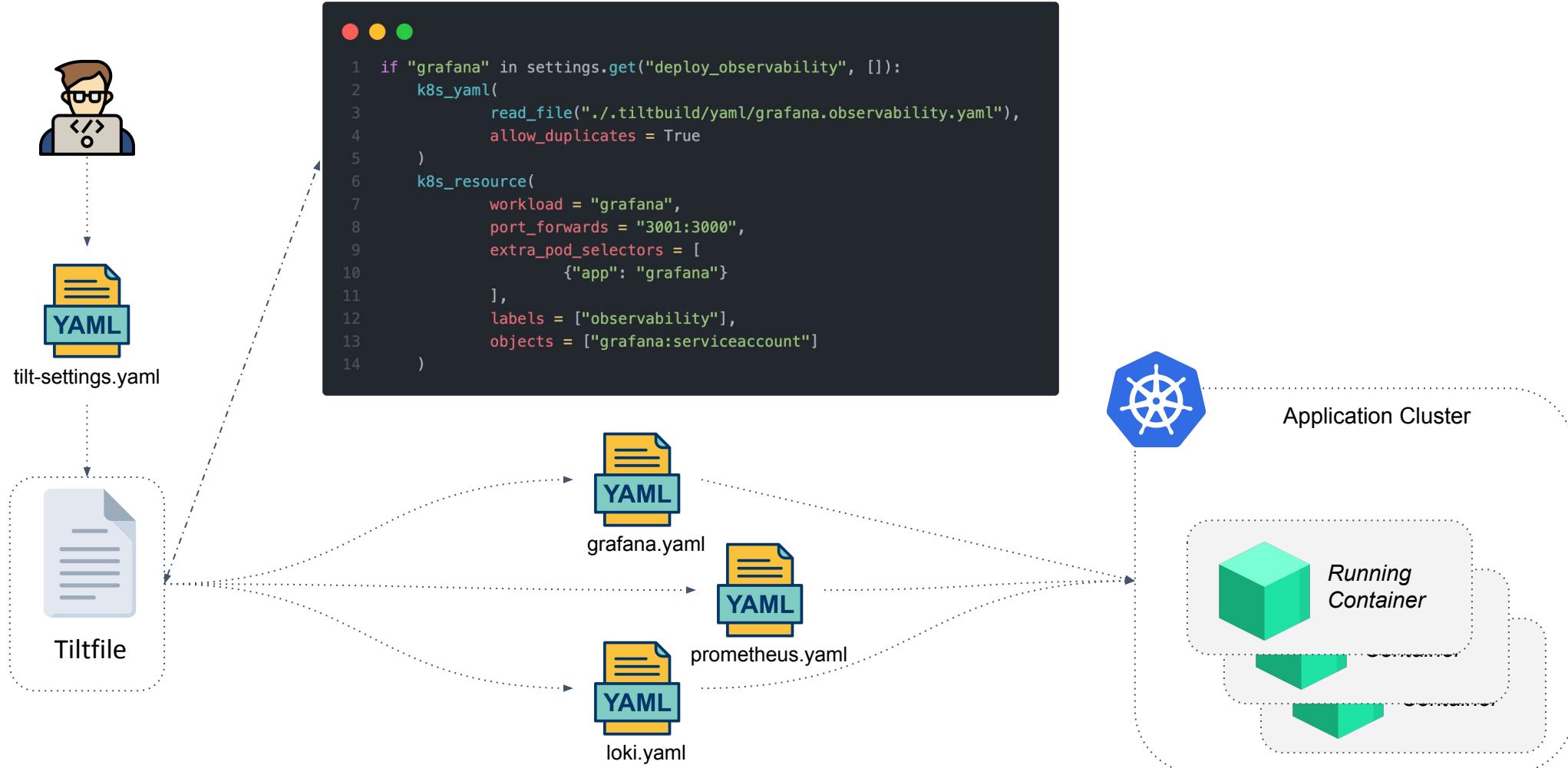
Europe 2023



Enable the observability tools on demand



Enable the observability tools on demand



What makes a good development workflow?

Finally...

What makes a good development workflow?

-  Easy spin up → *tilt up*
-  Quick feedback loop
-  Debuggable
-  Visibility

What makes a good development workflow?

-  Easy spin up → *tilt up*
-  Quick feedback loop → Live Update
-  Debuggable
-  Visibility

What makes a good development workflow?

-  Easy spin up → *tilt up*
-  Quick feedback loop → Live Update
-  Debuggable → Remote Debugger
-  Visibility

What makes a good development workflow?

-  Easy spin up → *tilt up*
-  Quick feedback loop → Live Update
-  Debuggable → Remote Debugger
-  Visibility → UI & Observability tools



KubeCon



CloudNativeCon

Europe 2023

Thank you!

