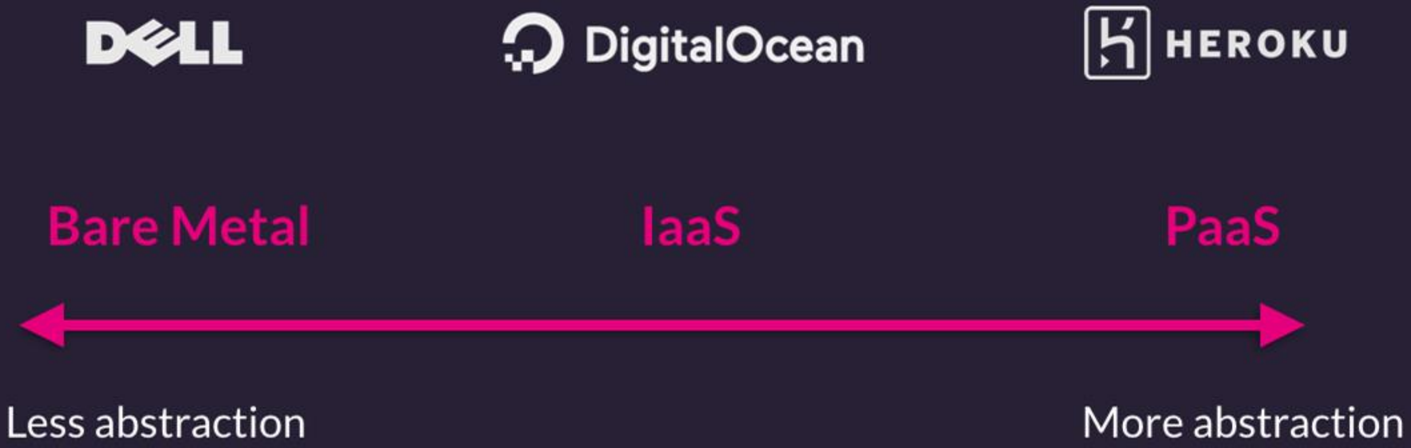
MOTHERSHIP

An Open-Source PaaS for Small Applications

Deployment Options Compared



DELL

DigitalOcean

HEROKU

Bare Metal

IaaS

PaaS

Less abstraction

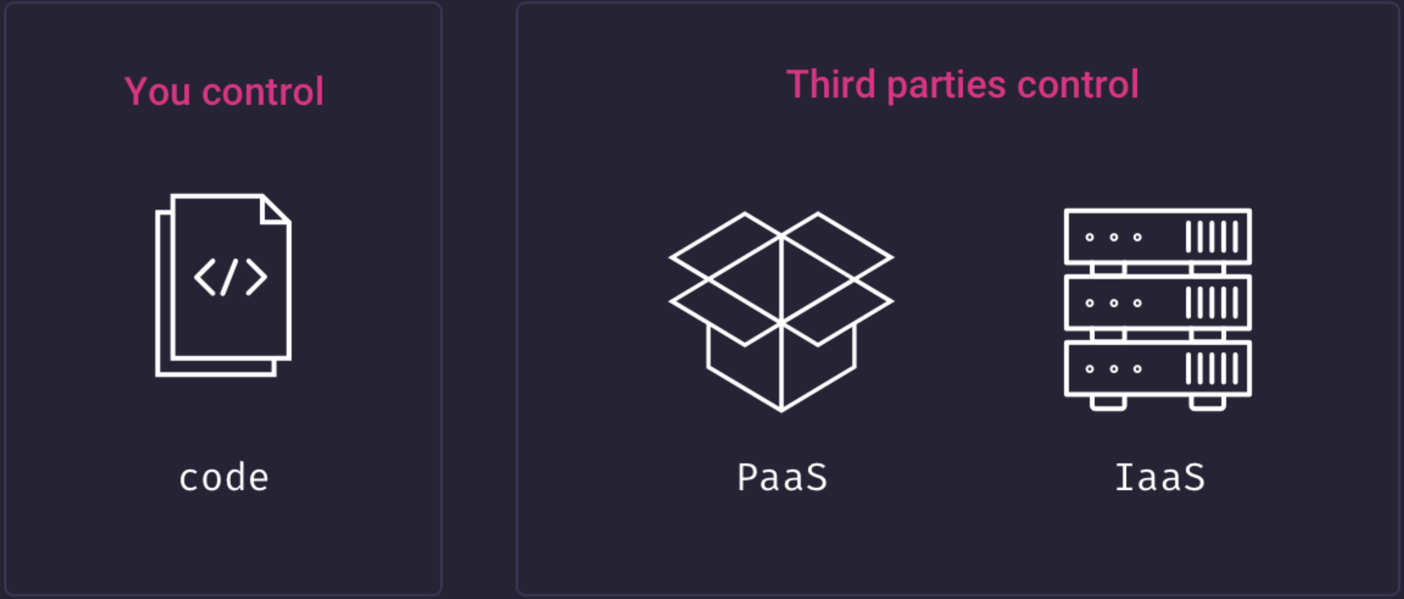
More abstraction

Deployment Options Compared

	Bare Metal	IaaS	PaaS
ABSTRACTION	none	virtual servers	app instances
CONCERNS	physical hardware, …up to app concerns	operating system, …up to app concerns	app concerns
USE-CASE	infrastructure is your core competency	fine-grained control of OS or host	just want to deploy apps

Traditional PaaS

PaaS is an intermediary between you and the underlying IaaS



You control

code

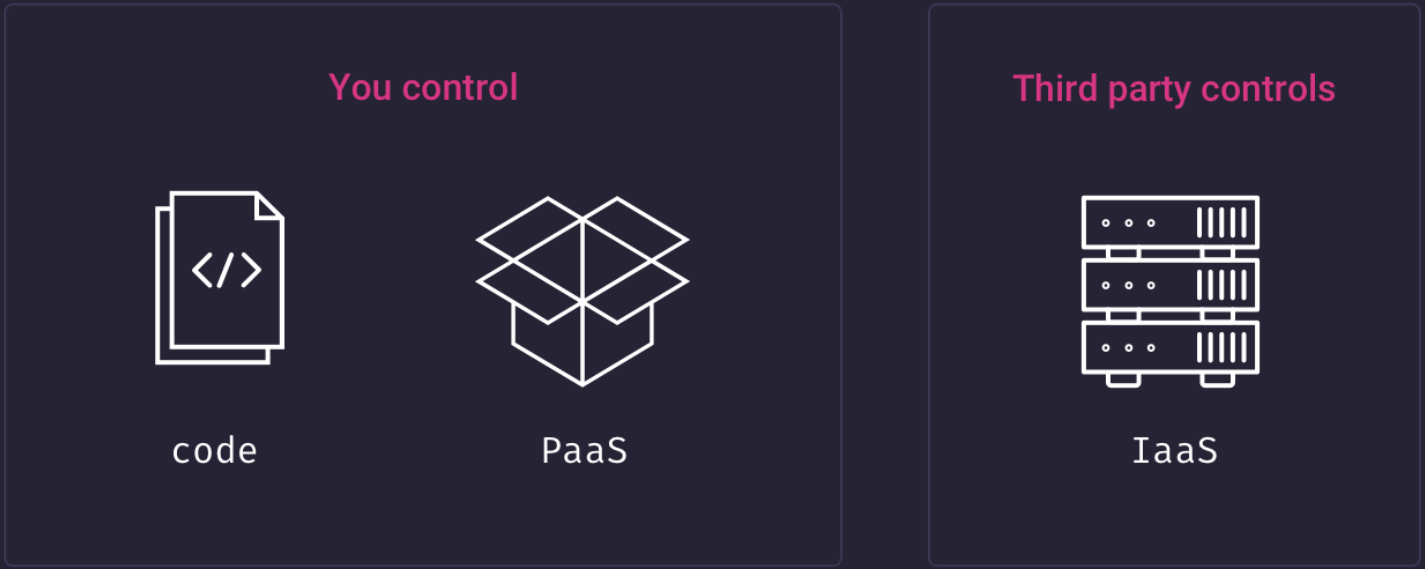
Third parties control

PaaS

IaaS

What's an Open-Source PaaS?

PaaS software that you clone and install directly on IaaS



Benefits of an Open-Source PaaS

- ***Control:*** only subject to one third-party's business decisions
- ***Provider Agnostic:*** switch IaaS providers as needed
- ***Trust:*** only one third-party accesses your code/data
- ***Other OSS benefits:*** audit, fork, or patch the PaaS itself

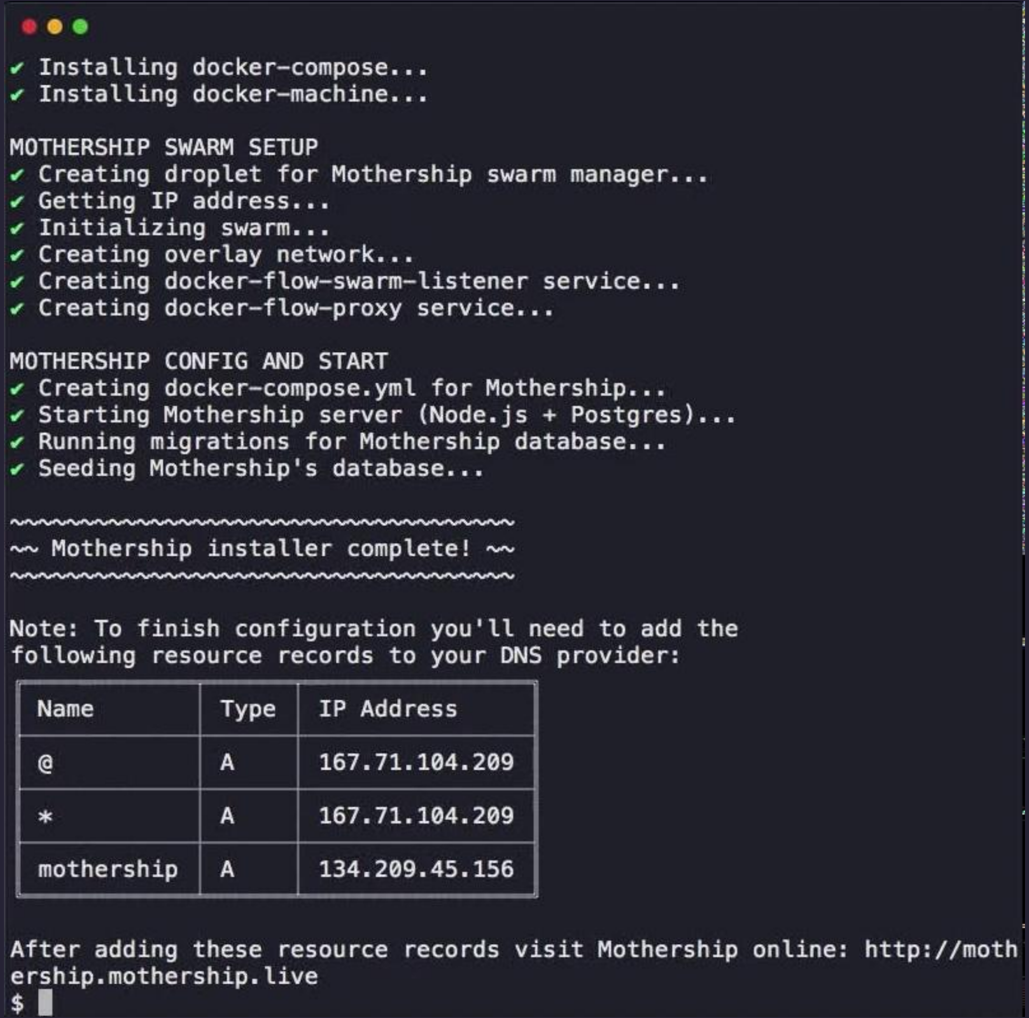
Goals for Deploying Apps on Mothership

- Deploy apps with no server/container knowledge
- Deploy from web or CLI
- Handle common languages with minimal configuration
- Centralized place to view and manage apps

Perfect for deploying internal company applications

Goals for Managing Mothership

- It should be easy to…
  - get your Mothership up and running
  - scale your Mothership as the number of apps grows



```
✔ Installing docker-compose...
✔ Installing docker-machine...

MOTHERSHIP SWARM SETUP
✔ Creating droplet for Mothership swarm manager...
✔ Getting IP address...
✔ Initializing swarm...
✔ Creating overlay network...
✔ Creating docker-flow-swarm-listener service...
✔ Creating docker-flow-proxy service...

MOTHERSHIP CONFIG AND START
✔ Creating docker-compose.yml for Mothership...
✔ Starting Mothership server (Node.js + Postgres)...
✔ Running migrations for Mothership database...
✔ Seeding Mothership's database...

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~ Mothership installer complete! ~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Note: To finish configuration you'll need to add the
following resource records to your DNS provider:
┌────────────┬──────┬────────────────┐
│ Name       │ Type │ IP Address     │
├────────────┼──────┼────────────────┤
│ @          │ A    │ 167.71.104.209 │
├────────────┼──────┼────────────────┤
│ *          │ A    │ 167.71.104.209 │
├────────────┼──────┼────────────────┤
│ mothership │ A    │ 134.209.45.156 │
└────────────┴──────┴────────────────┘

After adding these resource records visit Mothership online: http://moth
ership.mothership.live
$ 
```

How We Built It

Problems a PaaS Has To Solve

Tenancy

Where do apps live?

Application Packaging

How do we build and run an environment for an app?

Resource Scheduling

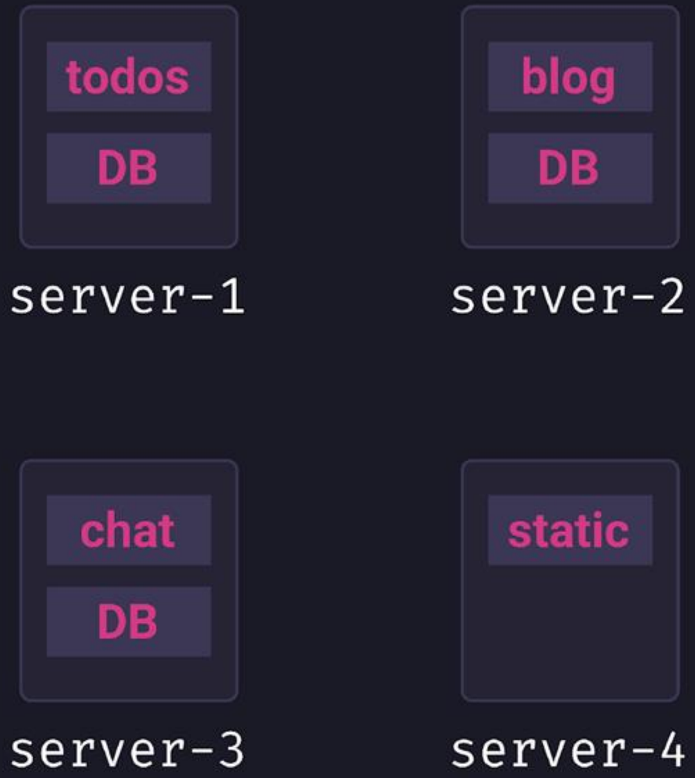
How do we keep track of deployed apps? Manage infrastructure?

Service Discovery

How do we map URLs to deployed apps?

Tenancy

Where do apps live?



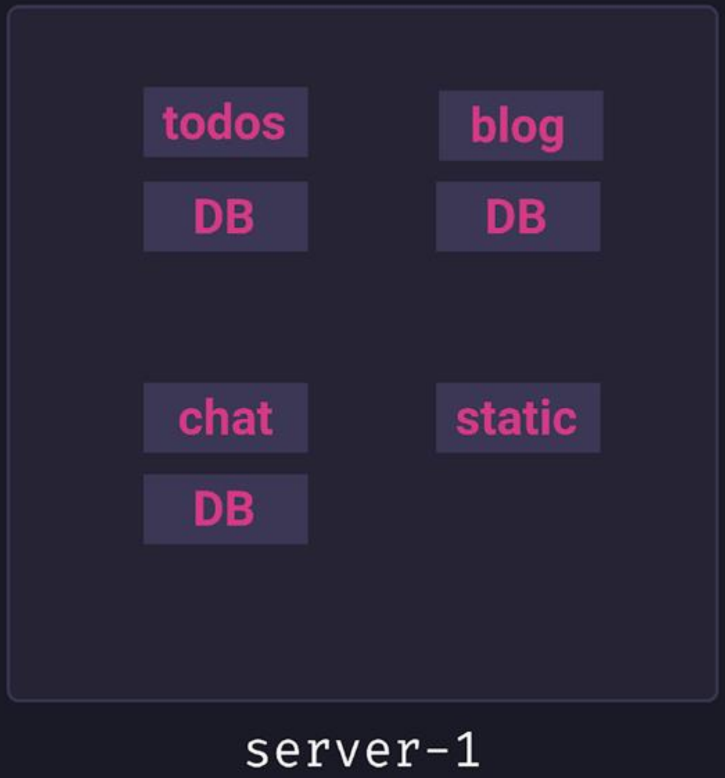
todos
DB
server-1

blog
DB
server-2

chat
DB
server-3

static
server-4

Single-tenancy



todos
DB

blog
DB

chat
DB

static

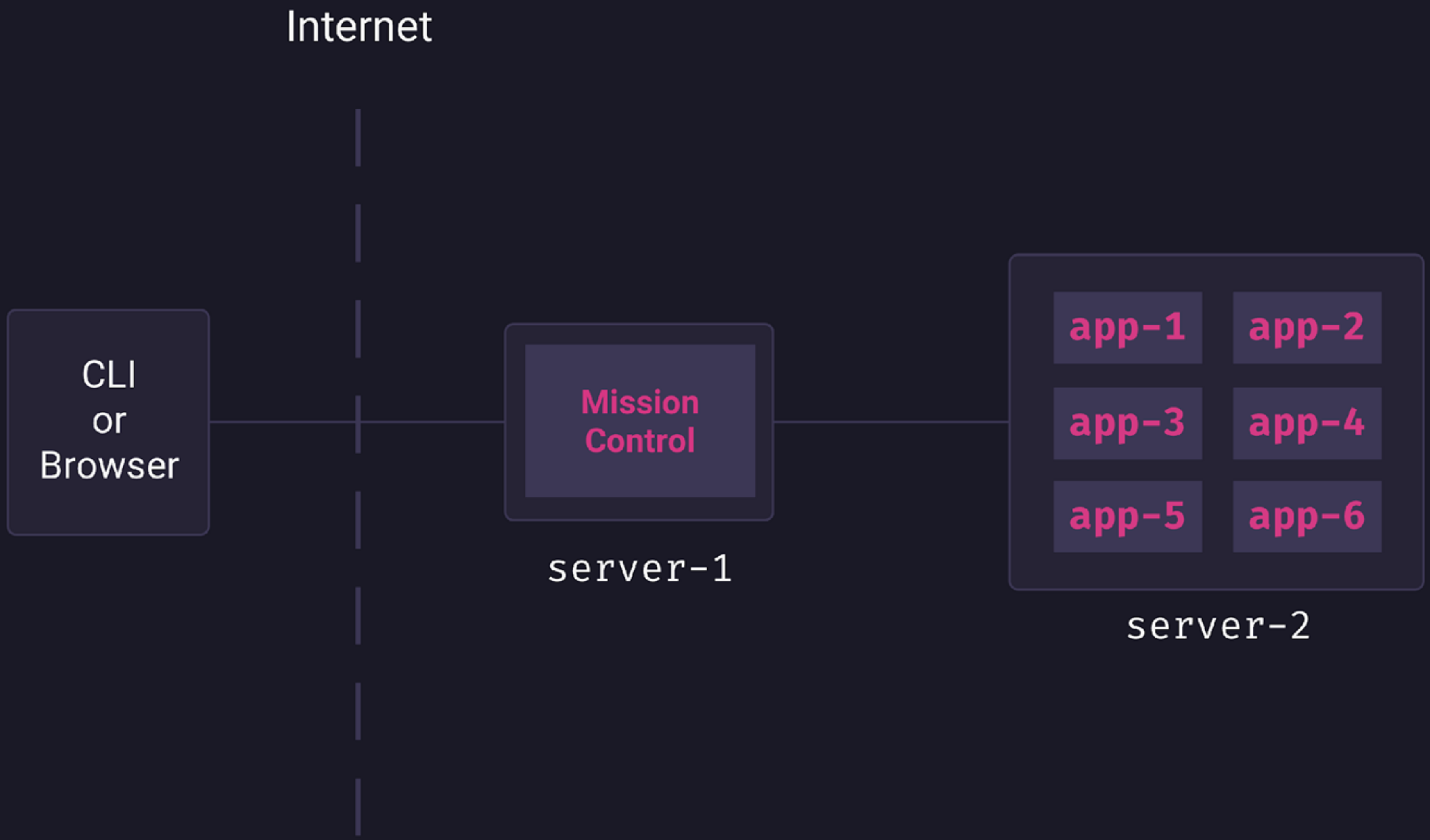
server-1

Multi-tenancy

Tenancy

	Single-tenancy	Multi-tenancy
DESCRIPTION	One app per server	Multiple apps per server
BENEFITS	• App can take full advantage of server • Run large apps • App isolation out of the box	• Run many, lighter apps on single host • Better resource utilization • Faster to create/delete apps

Multi-tenancy fits our use case of small, internal apps



Internet

CLI
or
Browser

Mission
Control

server-1

app-1
app-2
app-3
app-4
app-5
app-6

server-2

Packaging Applications

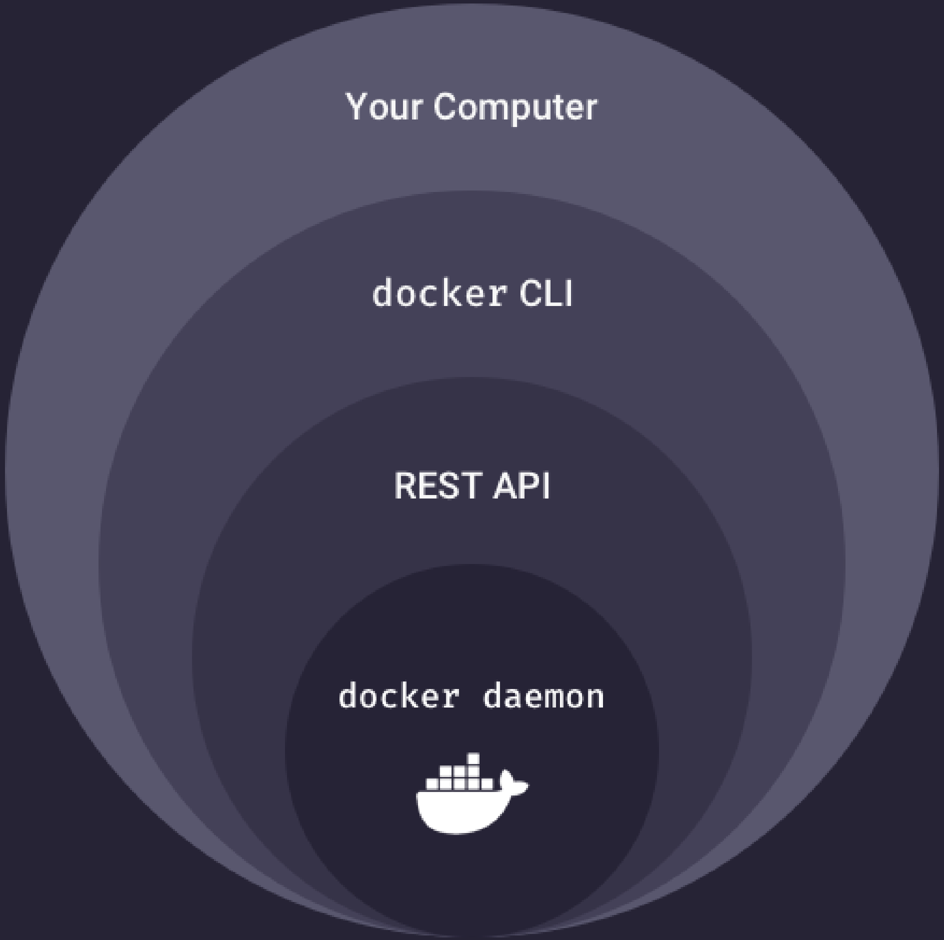
How do we create isolated environments where apps can run?

Containers

- Package up and execute code
- Isolated, lightweight, efficient
- Run many containers on same host
- From inside, appear to have their own OS
- We use Docker for containers

Docker Architecture

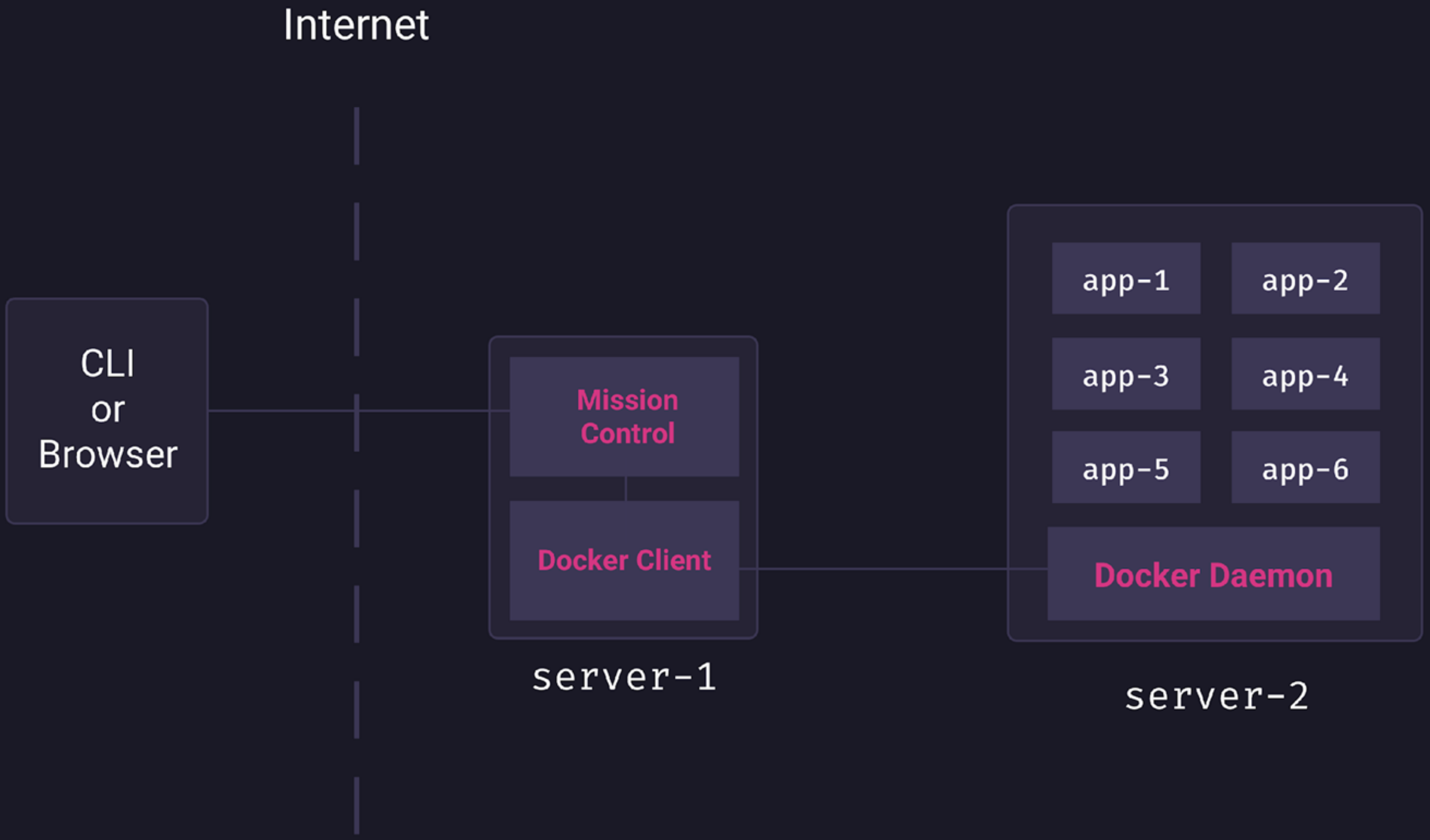
- **Docker Daemon** (server, heavy lifting happens here)
- **REST API** (for communicating with daemon)
- **Docker CLI** (most popular method for interacting with API)


Your Computer
docker CLI
REST API
docker daemon

Docker Architecture

> *“The Docker client and daemon can run on the same system, or you can connect a Docker client to a remote Docker daemon. The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface.”*

Docker Overview, docs.docker.com


Internet
CLI
or
Browser
Mission
Control
Docker Client
server-1
app-1
app-2
app-3
app-4
app-5
app-6
Docker Daemon
server-2

Packaging Applications: Part 2

How do we create containers specific to the app type?

Containerizing Arbitrary Apps

- How do we…
  - *install system-level dependencies?*
  - *install language-level dependencies?*
  - *determine/run correct command for starting app?*

Early Attempts

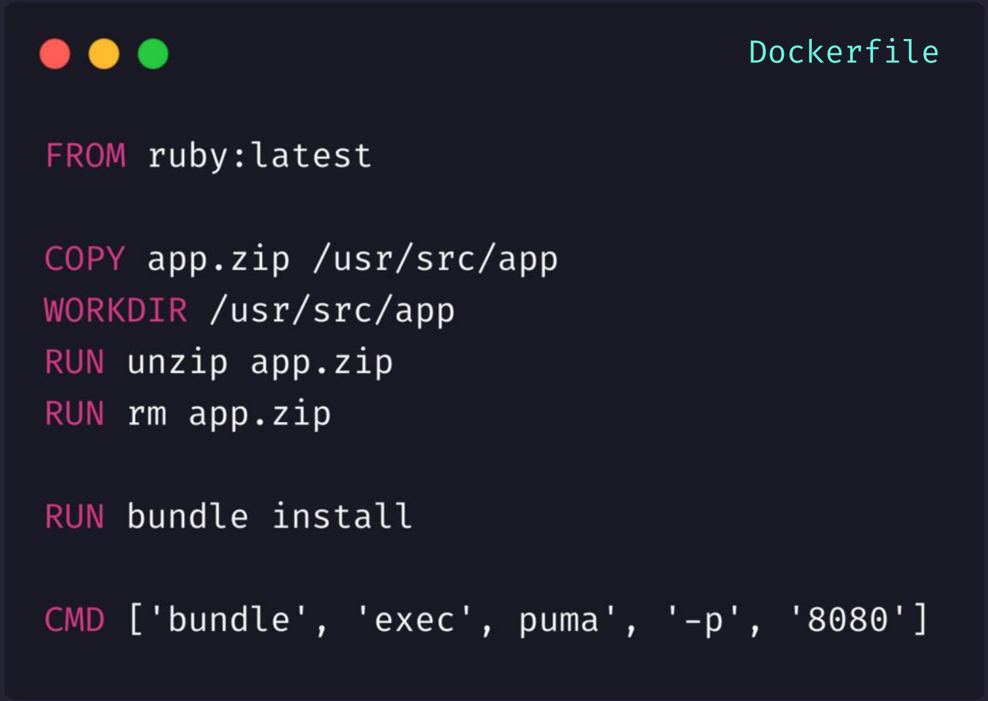
The earliest version of Mothership supported Rack-based Ruby apps…

Install latest version of Ruby

Copy source code into container

Run dependency installer

Start app



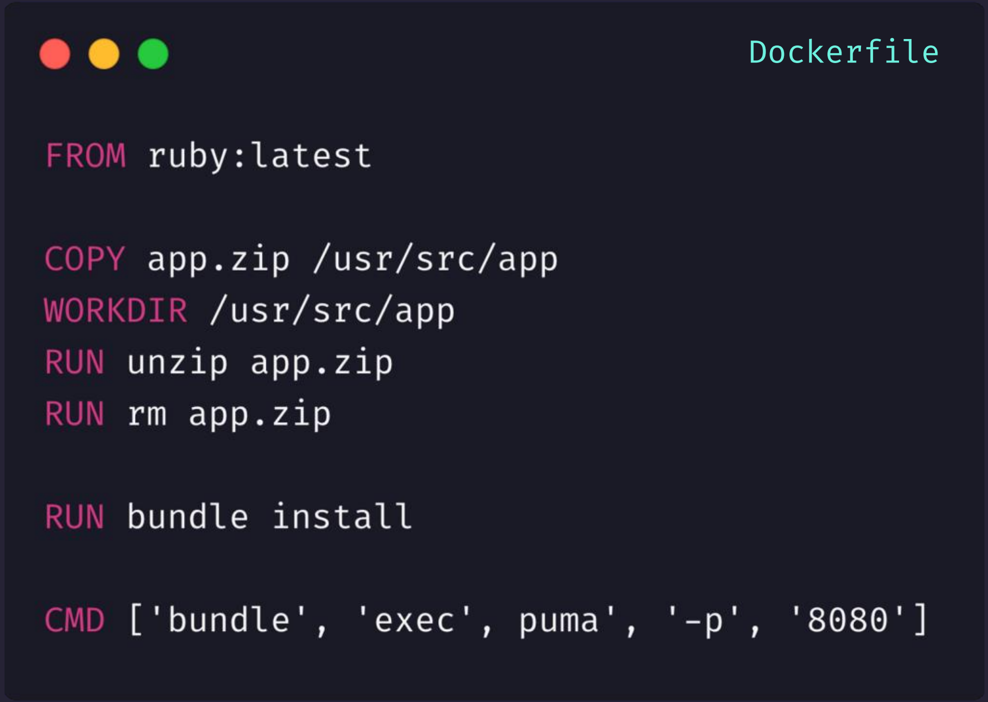
Dockerfile

```
FROM ruby:latest

COPY app.zip /usr/src/app
WORKDIR /usr/src/app
RUN unzip app.zip
RUN rm app.zip

RUN bundle install

CMD ['bundle', 'exec', puma', '-p', '8080']
```

Early Attempts

Problems:

- *Only supports Ruby*
- *Installs latest version of language* (your app might use something else)
- *Assumes a standardized start command*



```
FROM ruby:latest

COPY app.zip /usr/src/app
WORKDIR /usr/src/app
RUN unzip app.zip
RUN rm app.zip

RUN bundle install

CMD ['bundle', 'exec', puma', '-p', '8080']
```

Solution 1: Separate Dockerfiles

- One `Dockerfile` for each language
- Scan source code for language version
- Ask user for start command during deploy
- Insert values into generated `Dockerfile`

`Dockerfile-Node`

`Dockerfile-Python`

`Dockerfile-Ruby`

Solution 1: Separate Dockerfiles

Problems:

- *Scanning for language version imprecise*
- *Ask user for start command each deployment*
- *Dependency install edge cases, maintenance*

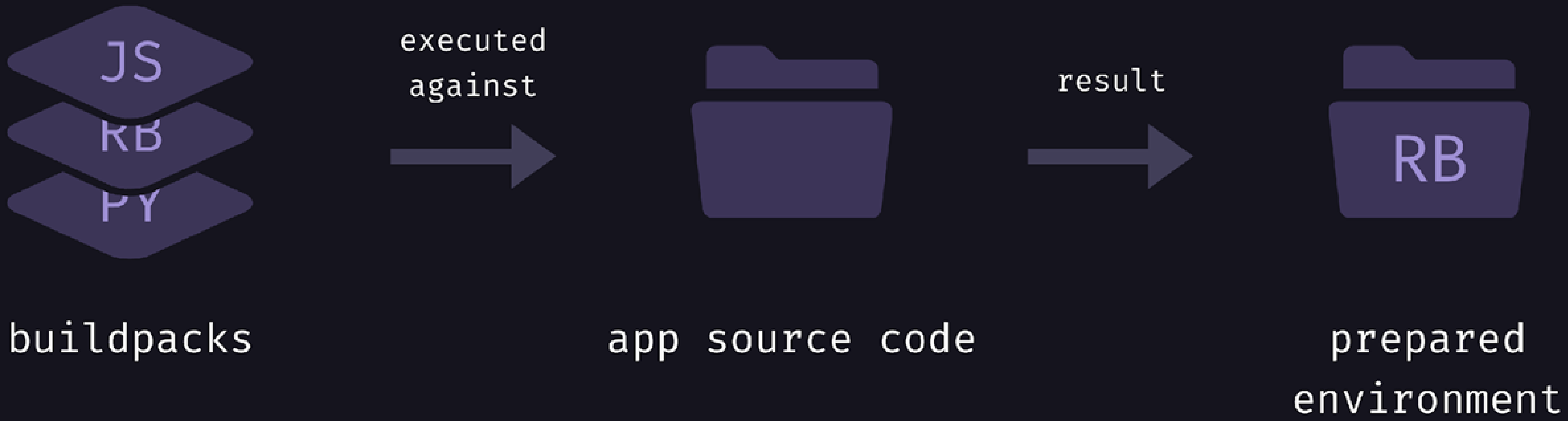
`Dockerfile-Node`

`Dockerfile-Python`

`Dockerfile-Ruby`

Solution 2: Buildpacks

- Popularized by Heroku
- Standardized instructions for creating app environment
- Install language and dependencies
- Detect start command from `Procfile`
- Many open-source, battle tested buildpacks available



buildpacks

executed against

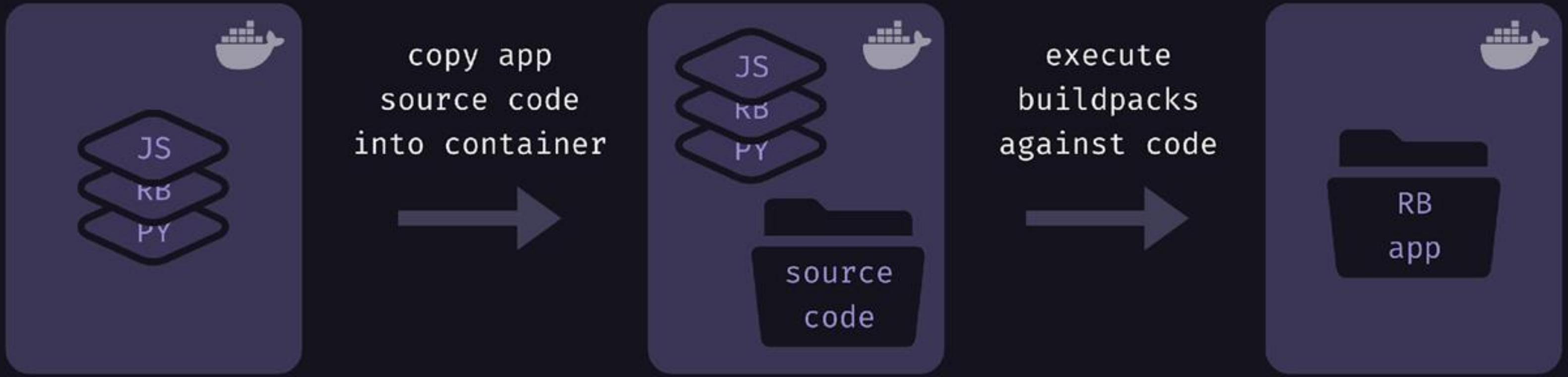
app source code

result

prepared environment

Buildpacks + Docker

- Start with base Docker image which includes buildpacks
- Copy app source code into container
- Execute buildpacks against app source code
- Save result as Docker container for app



docker container w/buildpacks installed

copy app source code into container

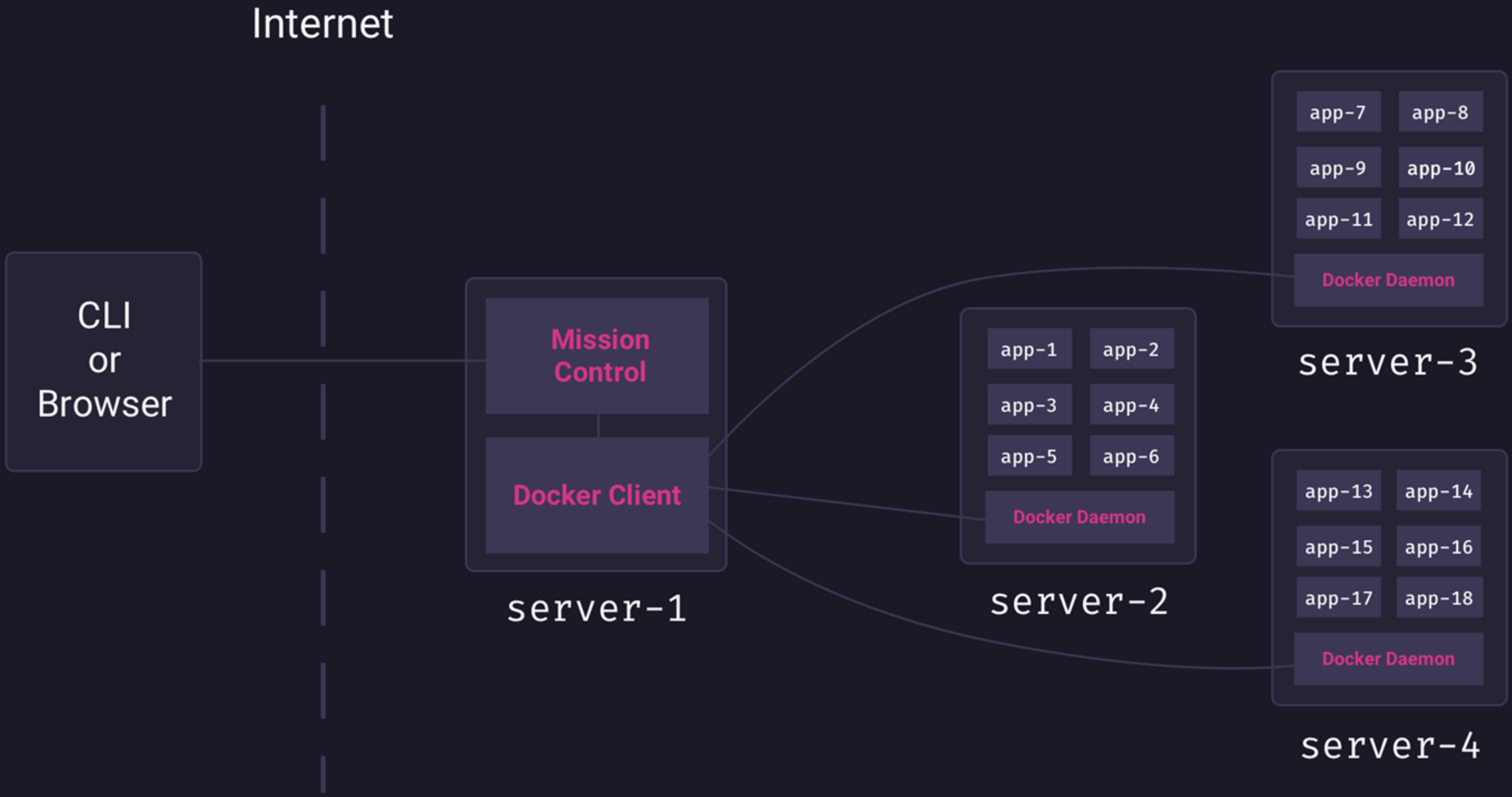
execute buildpacks against code

ready-to-run container

Resource Scheduling

Resource Scheduling

- We have a *single server* that runs *multiple apps*
- But… what happens if we need to run 20, 50, or 100 apps?
- We really need *multiple servers* that can run *multiple apps*



Internet

CLI
or
Browser

Mission Control

Docker Client

server-1

app-1 app-2
app-3 app-4
app-5 app-6

Docker Daemon

server-2

app-7 app-8
app-9 app-10
app-11 app-12

Docker Daemon

server-3

app-13 app-14
app-15 app-16
app-17 app-18

Docker Daemon

server-4

Resource Scheduling Problems

- *How do we know what containers are running?*
- *How do we know which server containers are running on?*
- *How do we know where to put new containers?*

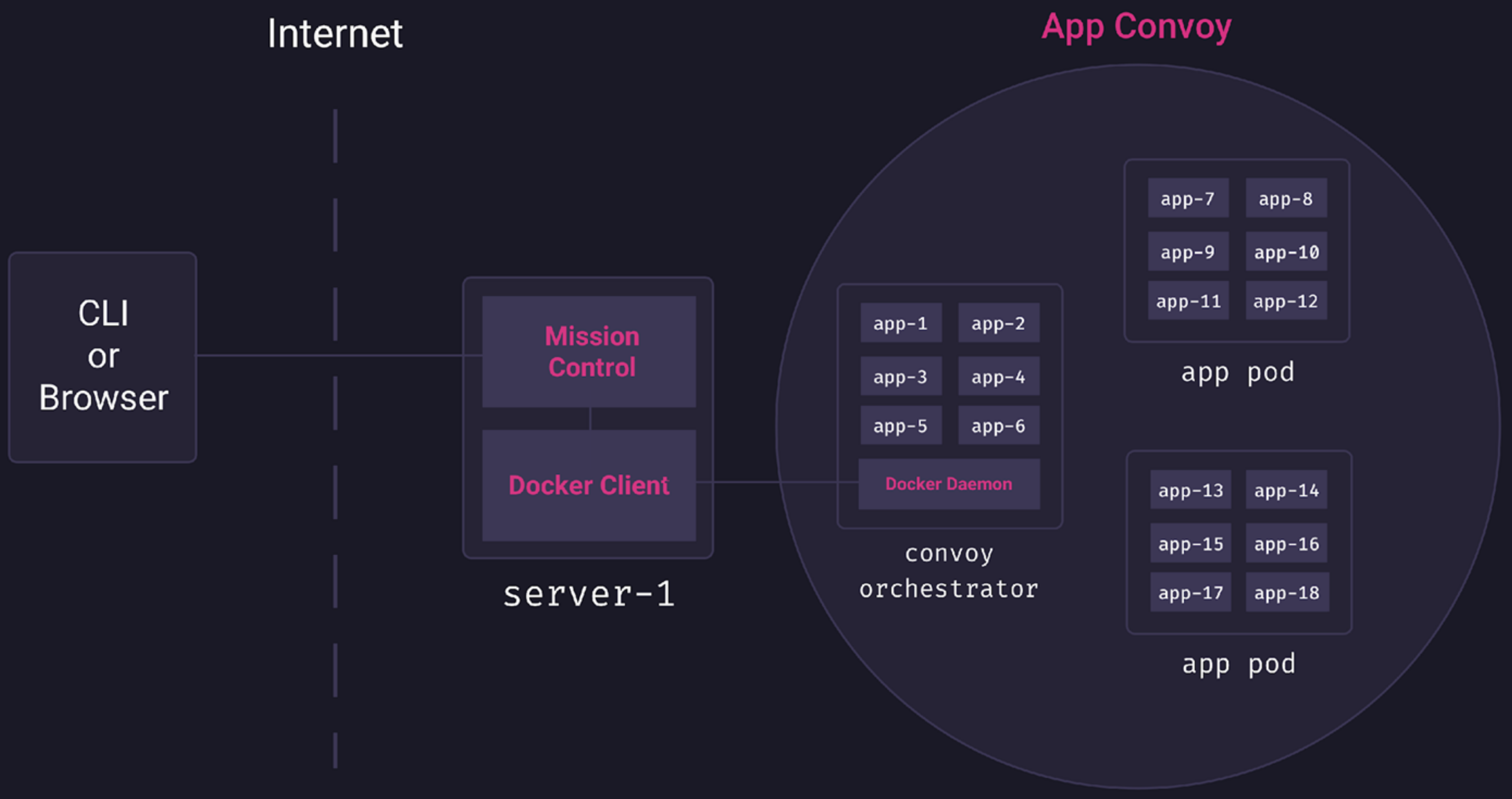
Container Orchestrators

Container Orchestrator Features

- Manage containers across *one or more* nodes (cluster)
- Interface for create/update/delete containers on a cluster
- Restart containers when needed
- Redistribute containers if nodes are added, removed, or fail

Container Orchestrator Basics

- **Cluster:** group of nodes (servers) working to run containers
- **Orchestrator/Manager:** node orchestrating the cluster
- **Workers/Nodes:** non-manager nodes running containers

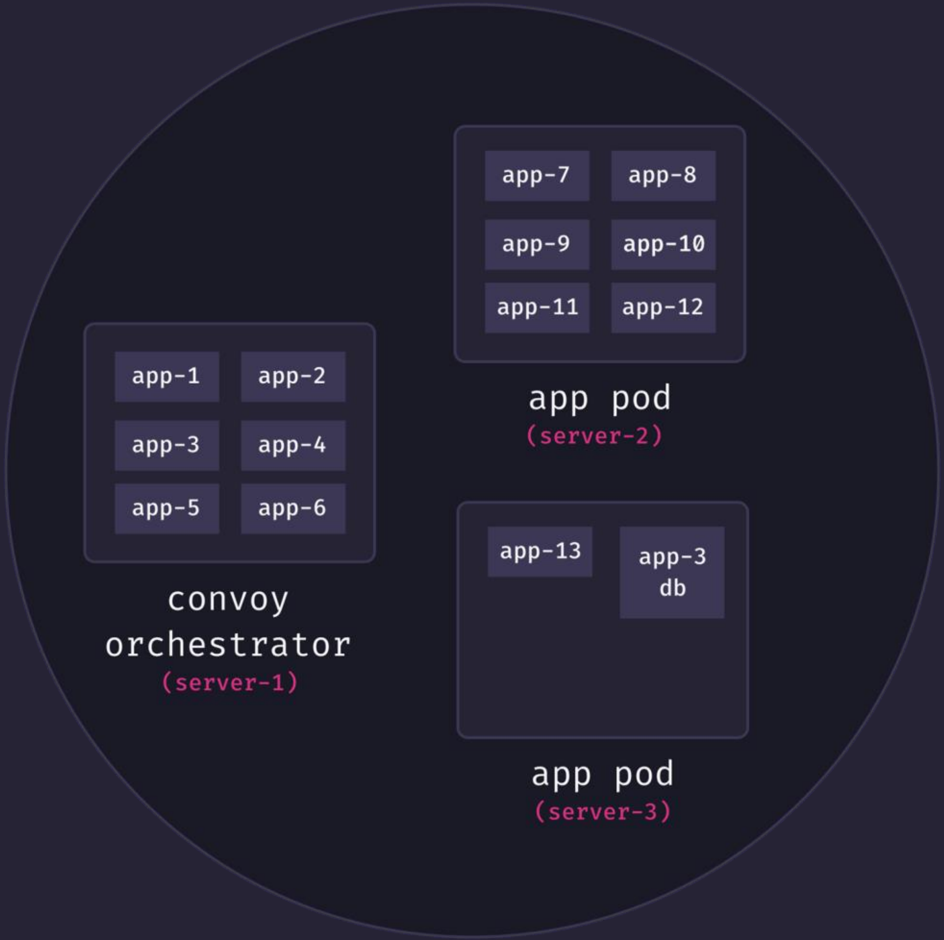

Internet
App Convoy
CLI
or
Browser
Mission Control
Docker Client
server-1
app-1
app-2
app-3
app-4
app-5
app-6
Docker Daemon
convoy orchestrator
app-7
app-8
app-9
app-10
app-11
app-12
app pod
app-13
app-14
app-15
app-16
app-17
app-18
app pod

Service Discovery

How do services talk to each other?

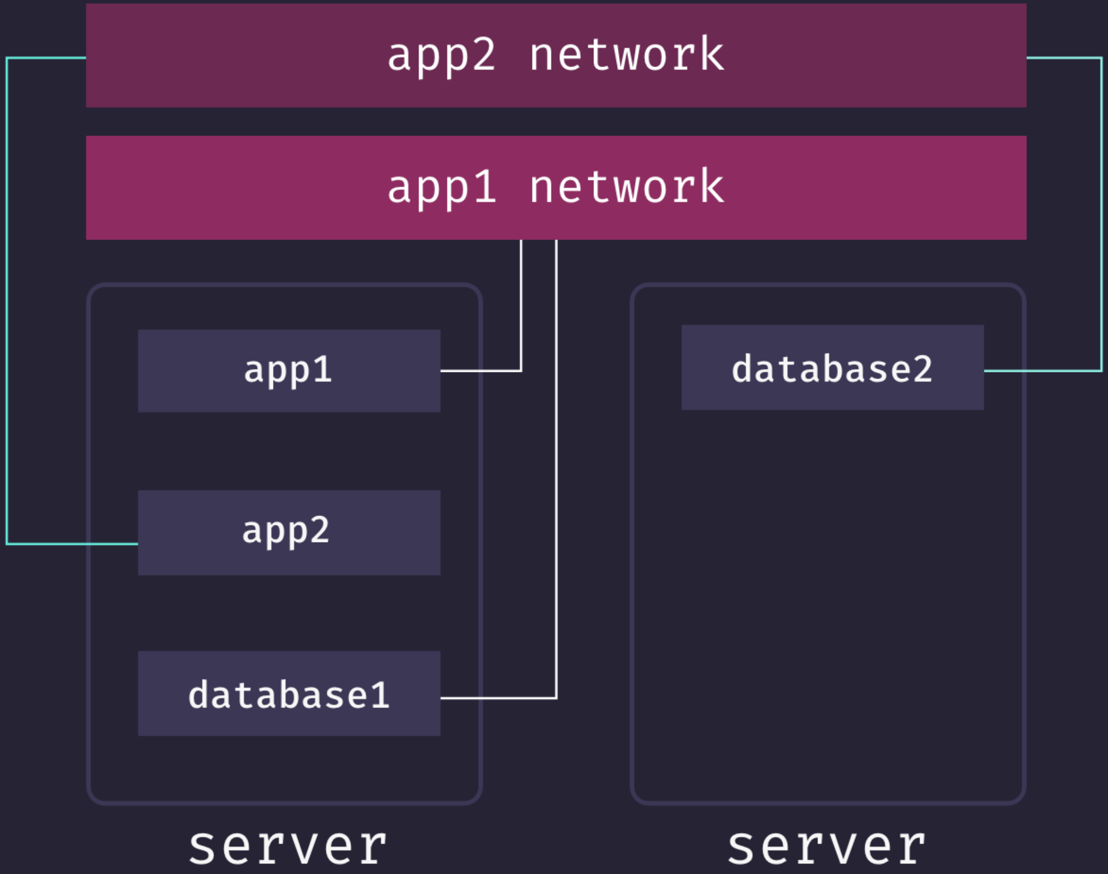
How do we map URLs to services?

App Convoy


app-1
app-2
app-3
app-4
app-5
app-6
convoy
orchestrator
(server-1)
app-7
app-8
app-9
app-10
app-11
app-12
app pod
(server-2)
app-13
app-3
db
app pod
(server-3)

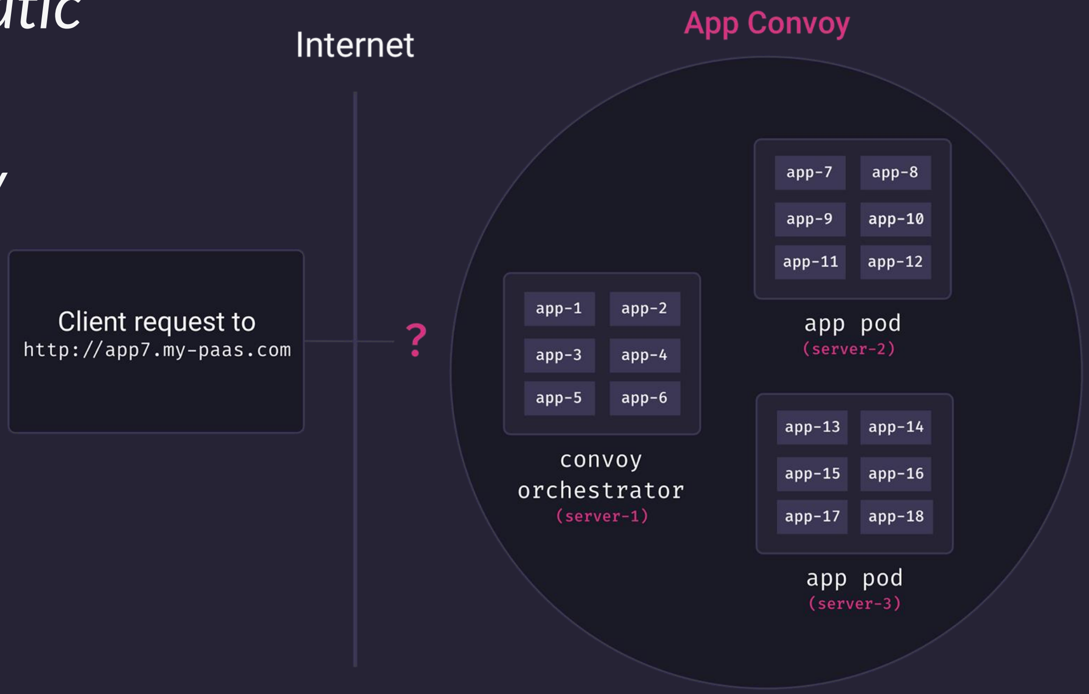
Inter-Service Communication

- How can containers on different machines communicate?
- Docker Overlay Networks
  - Distributed network spanning *multiple* Docker hosts
  - App and it’s accompanying DB container join network
  - Service names and DNS


app2 network
app1 network
app1
app2
database1
database2
server
server

Mapping URLs to Services

- *Apps don't have static IPs*
- *Apps can be on any node*
- *How to configure DNS?*

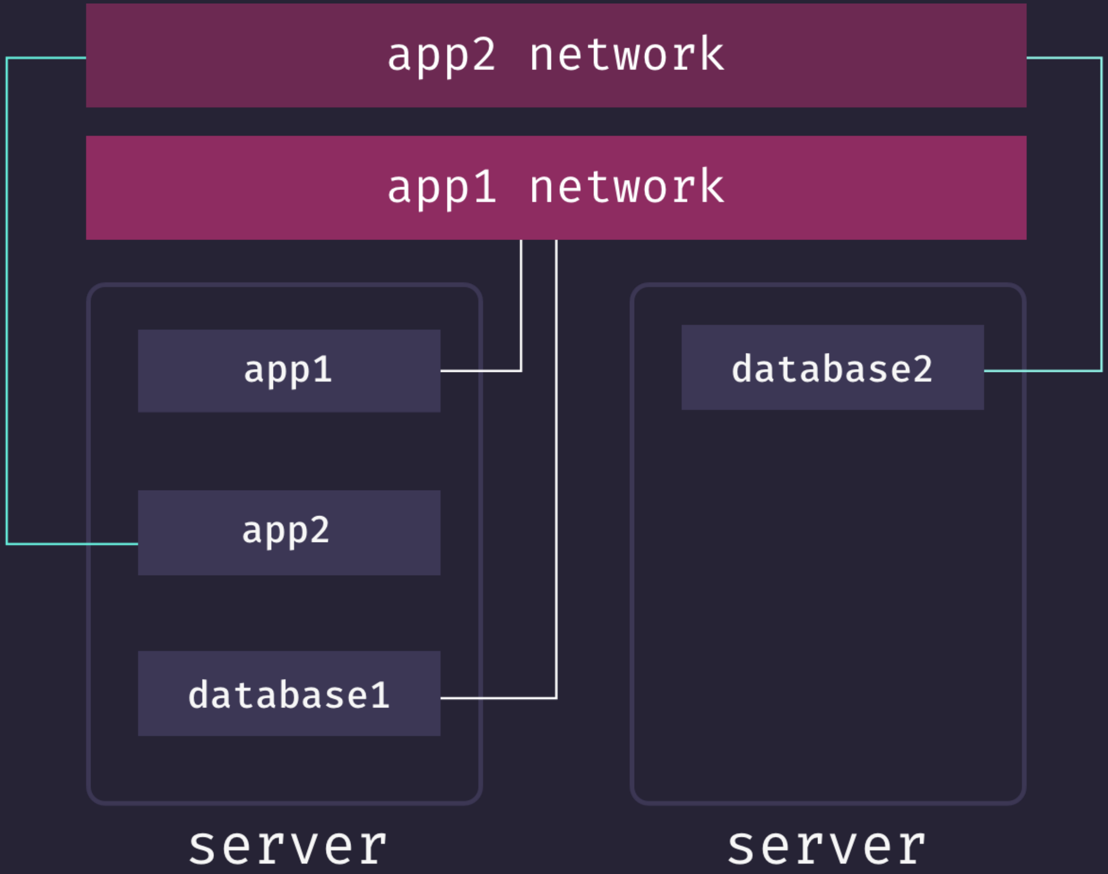


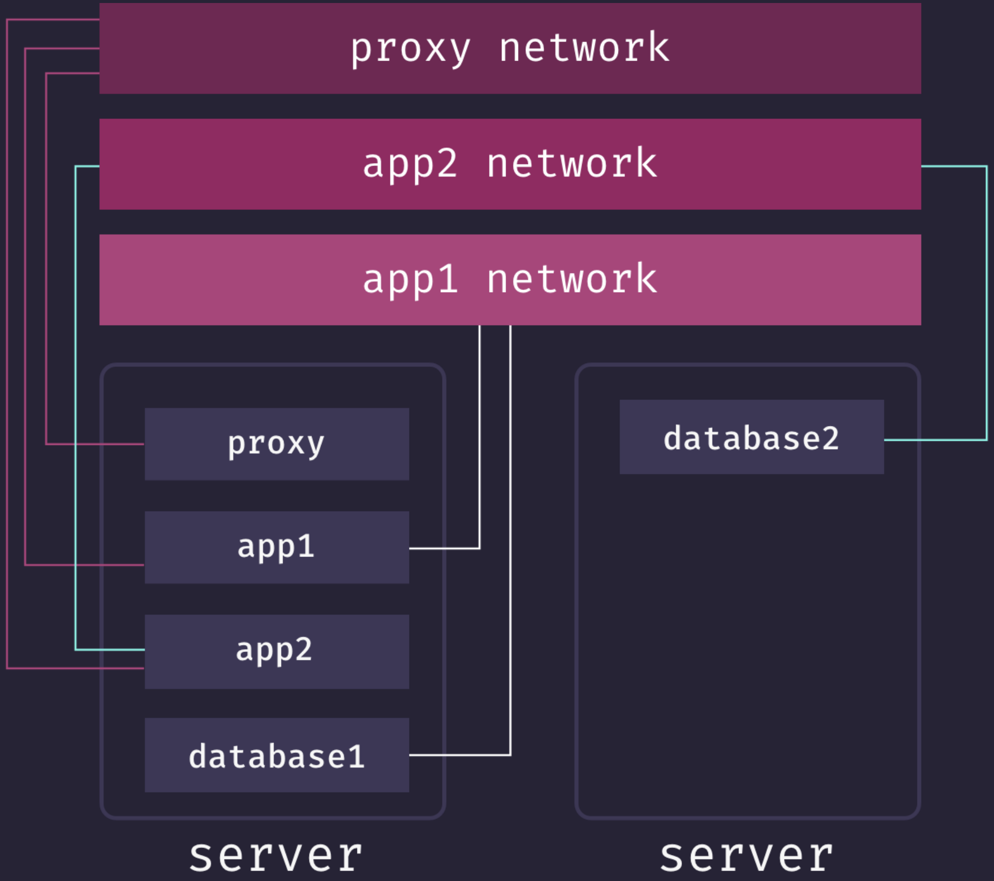
Ingress Routing Mesh

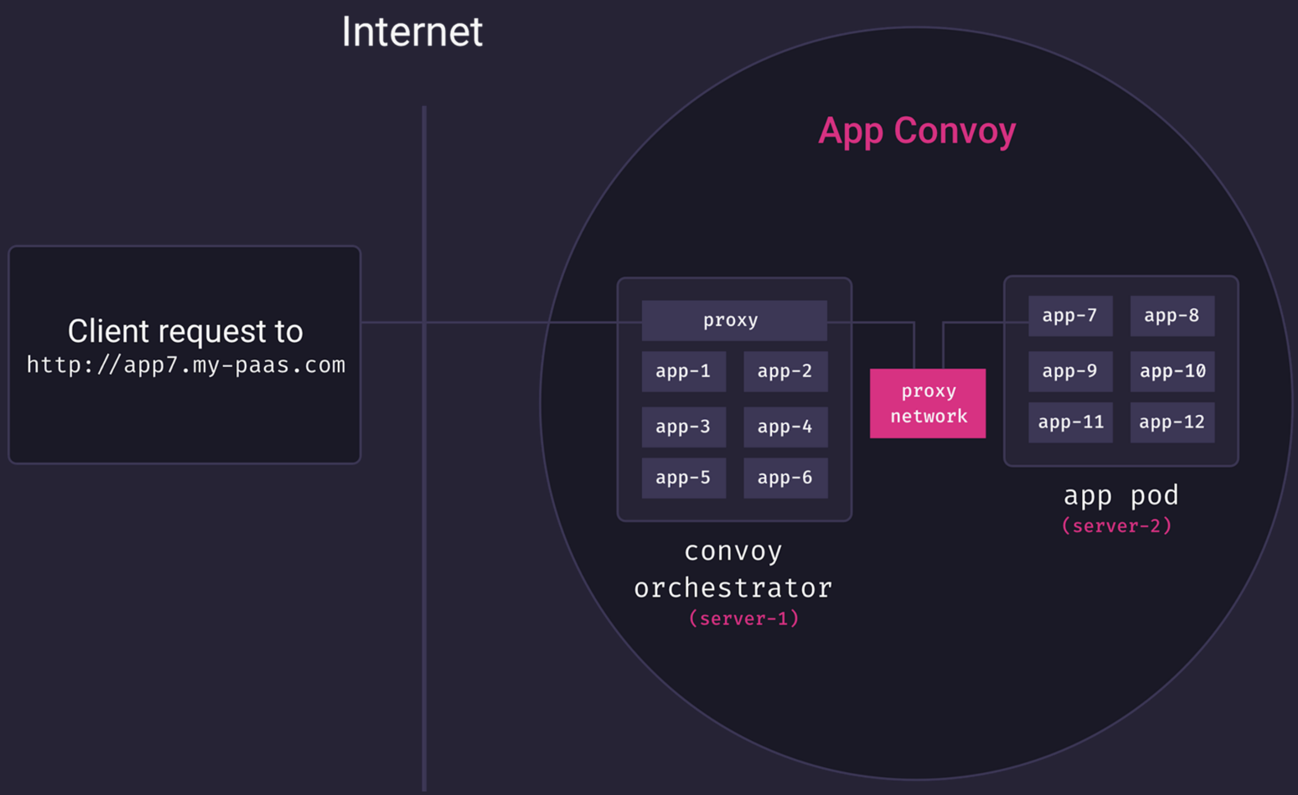
- Built into Docker
- Services publish unique port on node
- Layer 4 proxy/load balancer sends request to correct node
- URLs would have to look like this:
  - `https://my-paas.com:13541` → my-app
  - `https://my-paas.com:97654` → my-other-app
- Not what we're looking for!

Reverse Proxy Service

- L7 reverse proxy (HAProxy, Nginx) can route based on hostname
- How will we map hostnames to containers?
  - Container IPs are ephemeral
  - Containers can move between pods
- Another overlay network to the rescue!


app2 network
app1 network
app1
app2
database1
database2
server
server


proxy network
app2 network
app1 network
proxy
app1
app2
database1
database2
server
server



- *.my-paas.com → orchestrator IP
- Proxy service listens on port 80 via ingress routing mesh

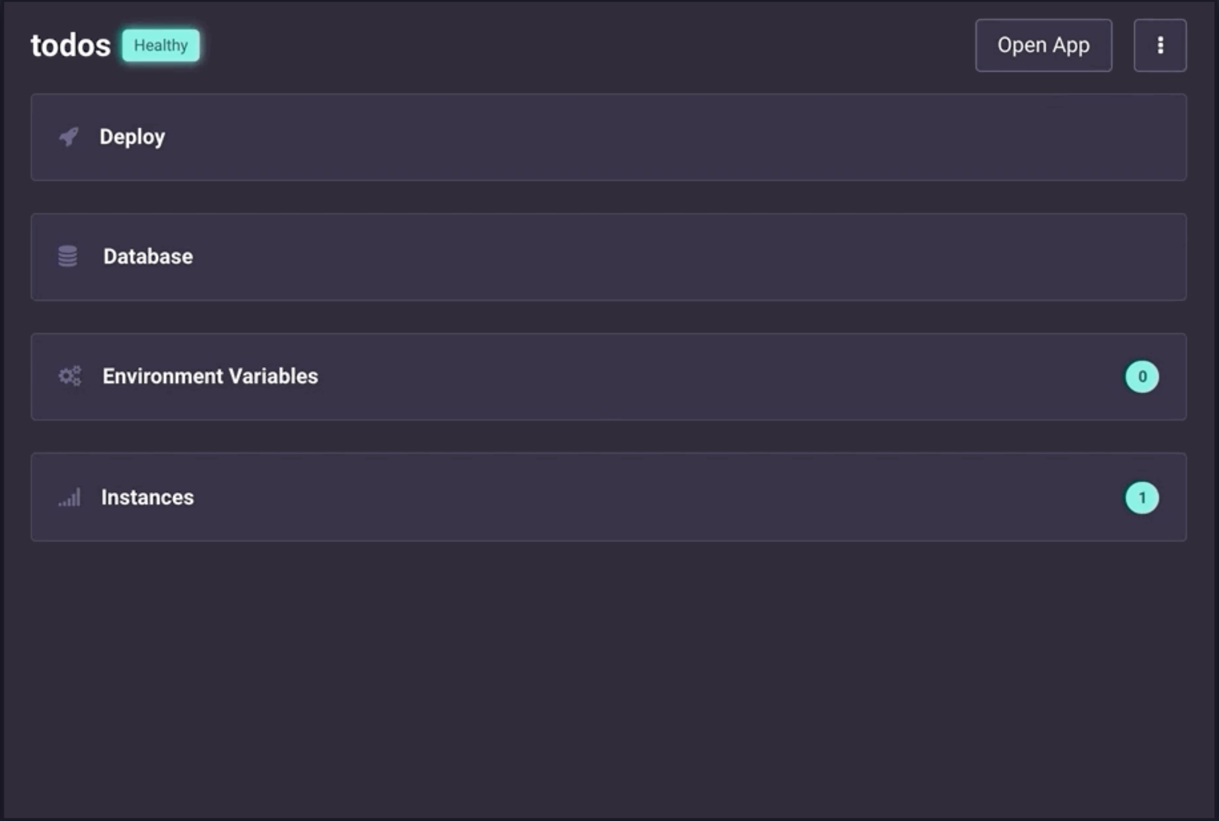
- Proxy routes request to app via app name through proxy overlay network

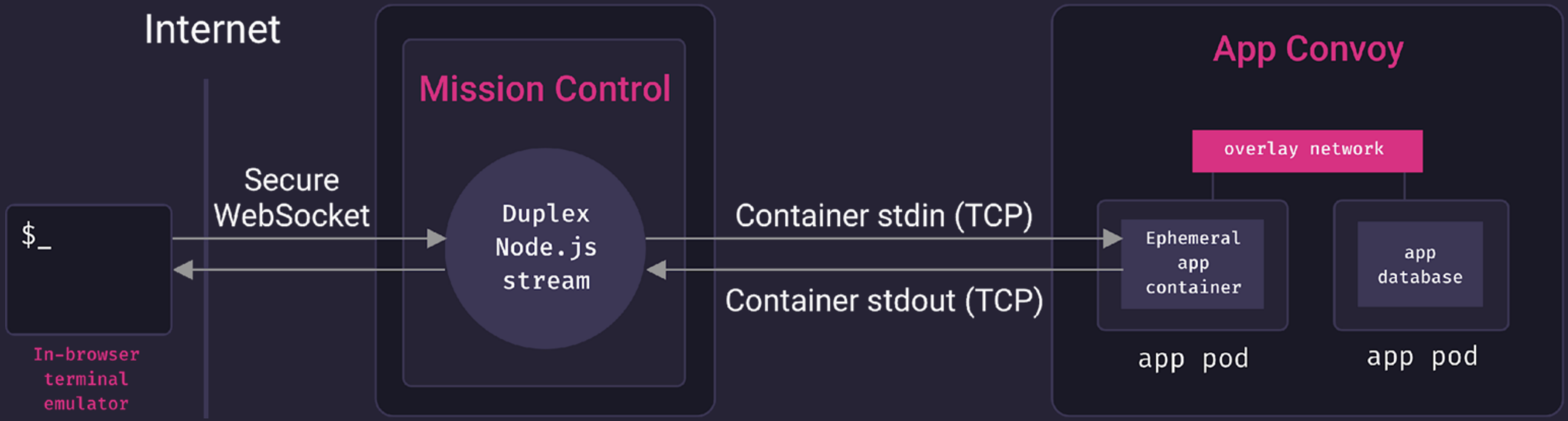
Implementing Essential Features

Essential Features

- Running terminal commands
- Database backups
- Health checks
- App service logs
- Build logs

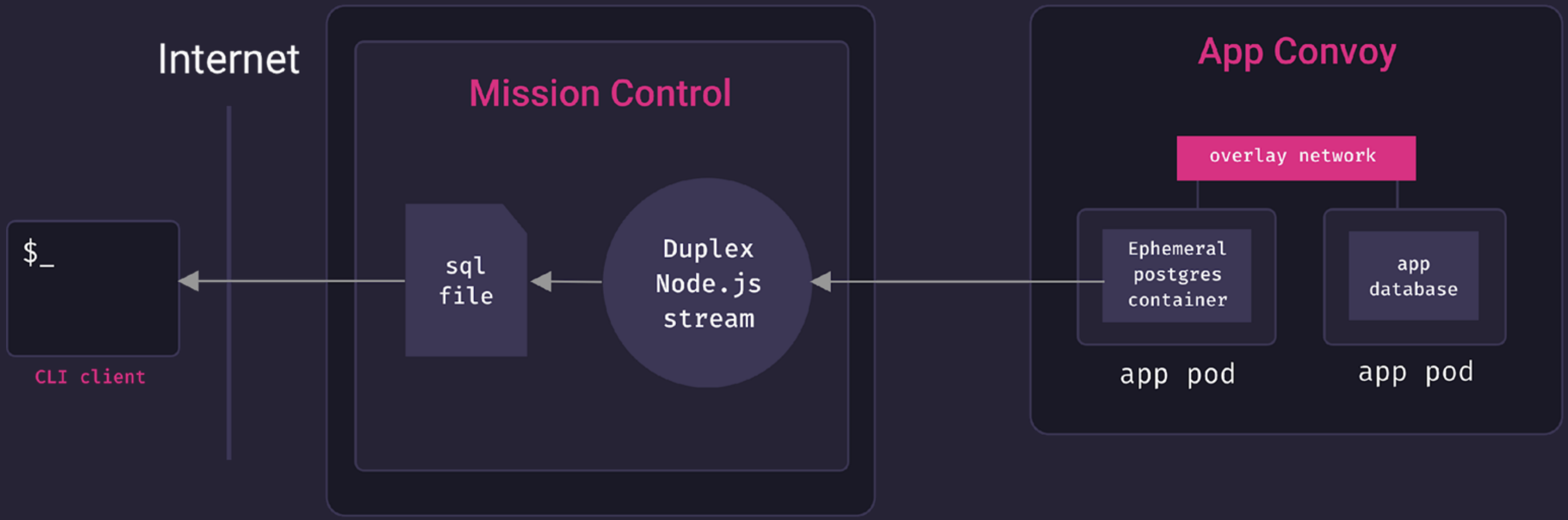
Running terminal commands



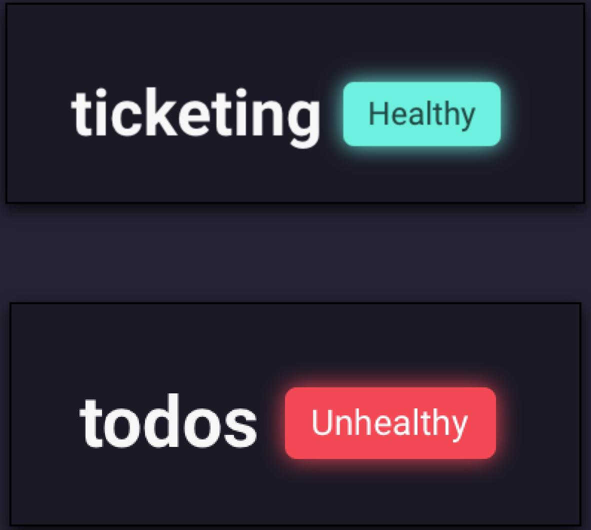

Internet
Mission Control
App Convoy
overlay network
Secure
WebSocket
$_
Duplex
Node.js
stream
Container stdin (TCP)
Container stdout (TCP)
Ephemeral
app
container
app
database
In-browser
terminal
emulator
app pod
app pod

Database Backups

```
$
```

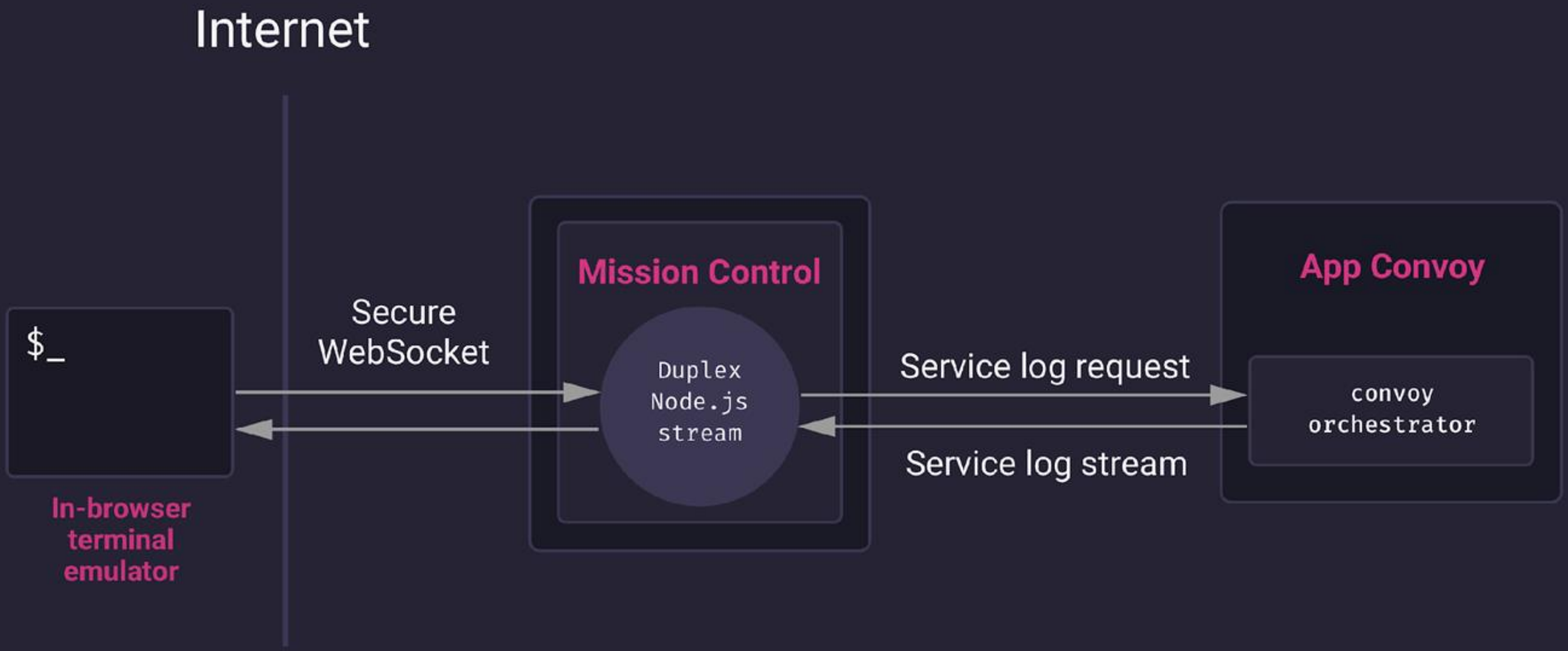

Internet
Mission Control
App Convoy
overlay network
$_
CLI client
sql file
Duplex Node.js stream
Ephemeral postgres container
app database
app pod
app pod

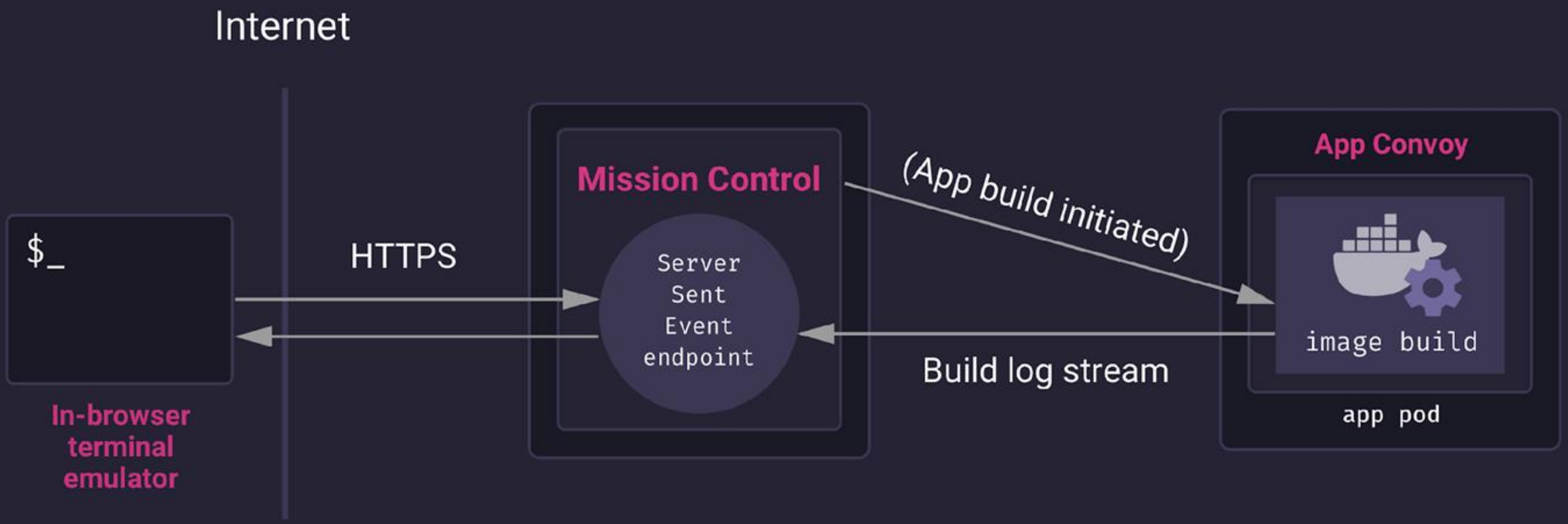
Health checks



ticketing Healthy

todos Unhealthy


Internet
$_
In-browser terminal emulator
Secure WebSocket
Mission Control
Duplex Node.js stream
Service log request
Service log stream
App Convoy
convoy orchestrator


Internet
$_
In-browser terminal emulator
HTTPS
Mission Control
Server Sent Event endpoint
(App build initiated)
Build log stream
App Convoy
image build
app pod

Future Work

- Testing
- Versioning and rollbacks
- Background jobs for apps
- CLI feature parity with web
- Migrate automated setup to Terraform

Team Mothership

Ian Evans

New York, NY

Jon Kulton

Akron, OH

Bharat Agarwal

Bangalore, India