Replit end to end

Beginner to advance

Giving some Context

Pre requisites

1. Basic Coding (loops, if else, variables)
2. Node.js

Good to haves

1. Docker / Containerization
2. Kubernetes
3. AWS ASGs

What we'll learn

Basic

1. Backend communication
2. Docker / Containerization
3. Isolated environments
4. Remote code execution
5. repl.it system design/architecture

Advance

1. Kubernetes
2. Pseudo Terminals
3. Nix

Before we start - Disclaimer

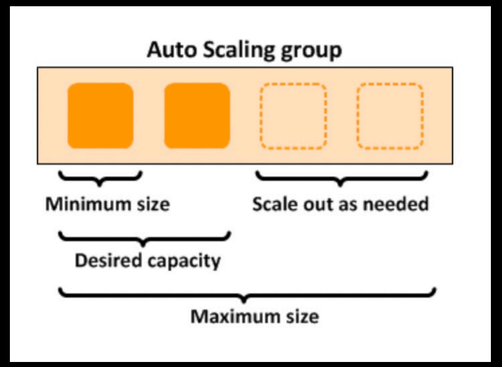
We'll be taking 3 approaches to solve this problem

Beginner friendly

1. How I would implement it from first principles back in college
2. Highly insecure
3. If you use this approach in an interview you'll be rejected
4. Good to know what's happening though
5. Doesn't scale

Cloud specific Autoscaling constructs

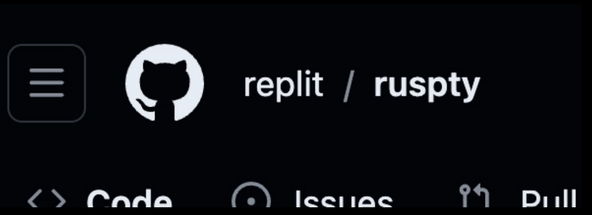
1. Mid approach
2. Great for a new startup
3. Secure, autoscales
4. Uses cloud dependant constructs (ASG, ECS)



Cloud native Approach

1. Good approach
2. Autoscales, secure
3. How I would build replit

How did I think of this architecture?



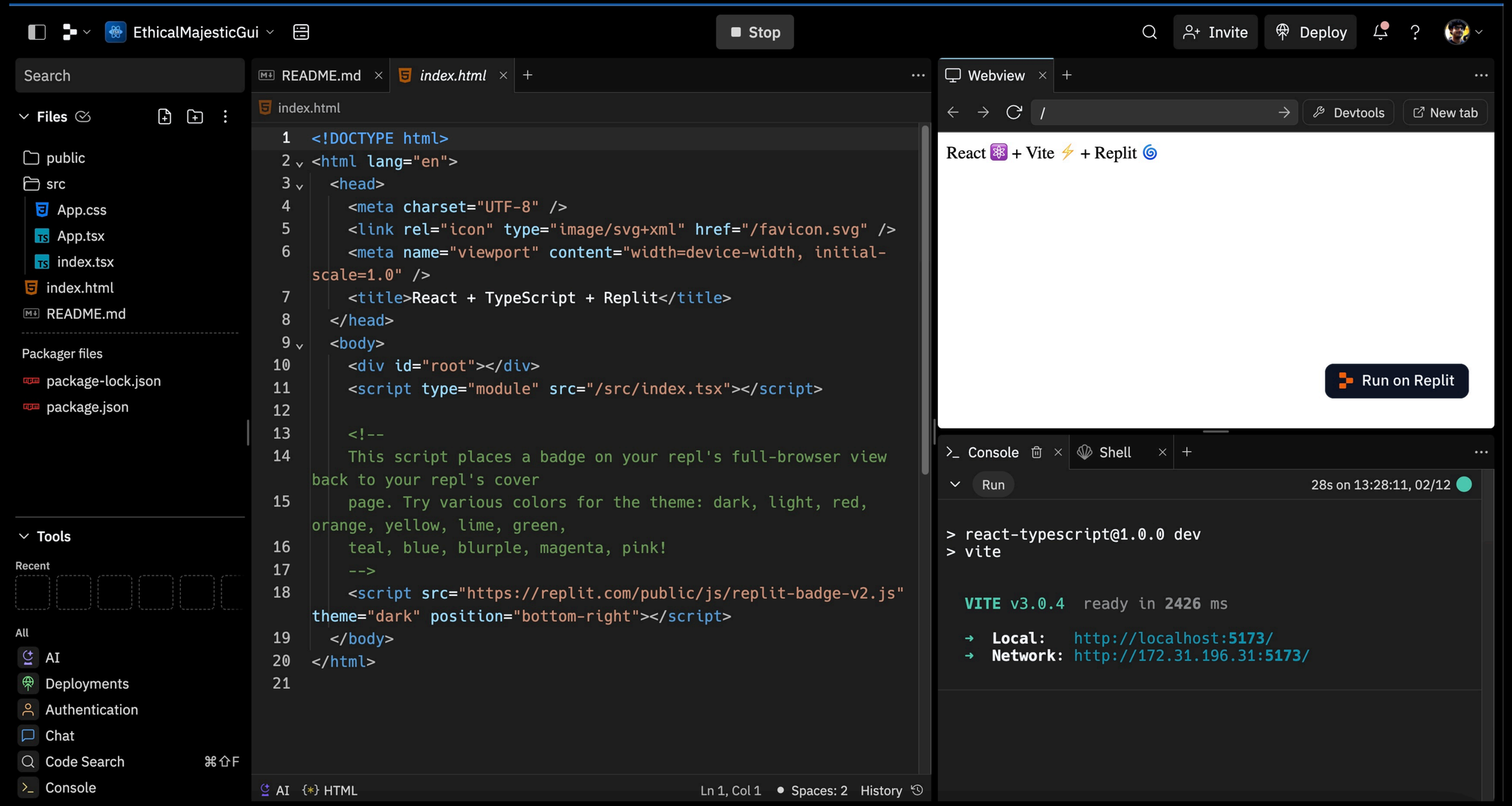
First principles (~8 years of coding)
Reverse engineering repl.it (~3 hours)
Going through their bog posts, GitHub and videos on YT

All are linked in the description

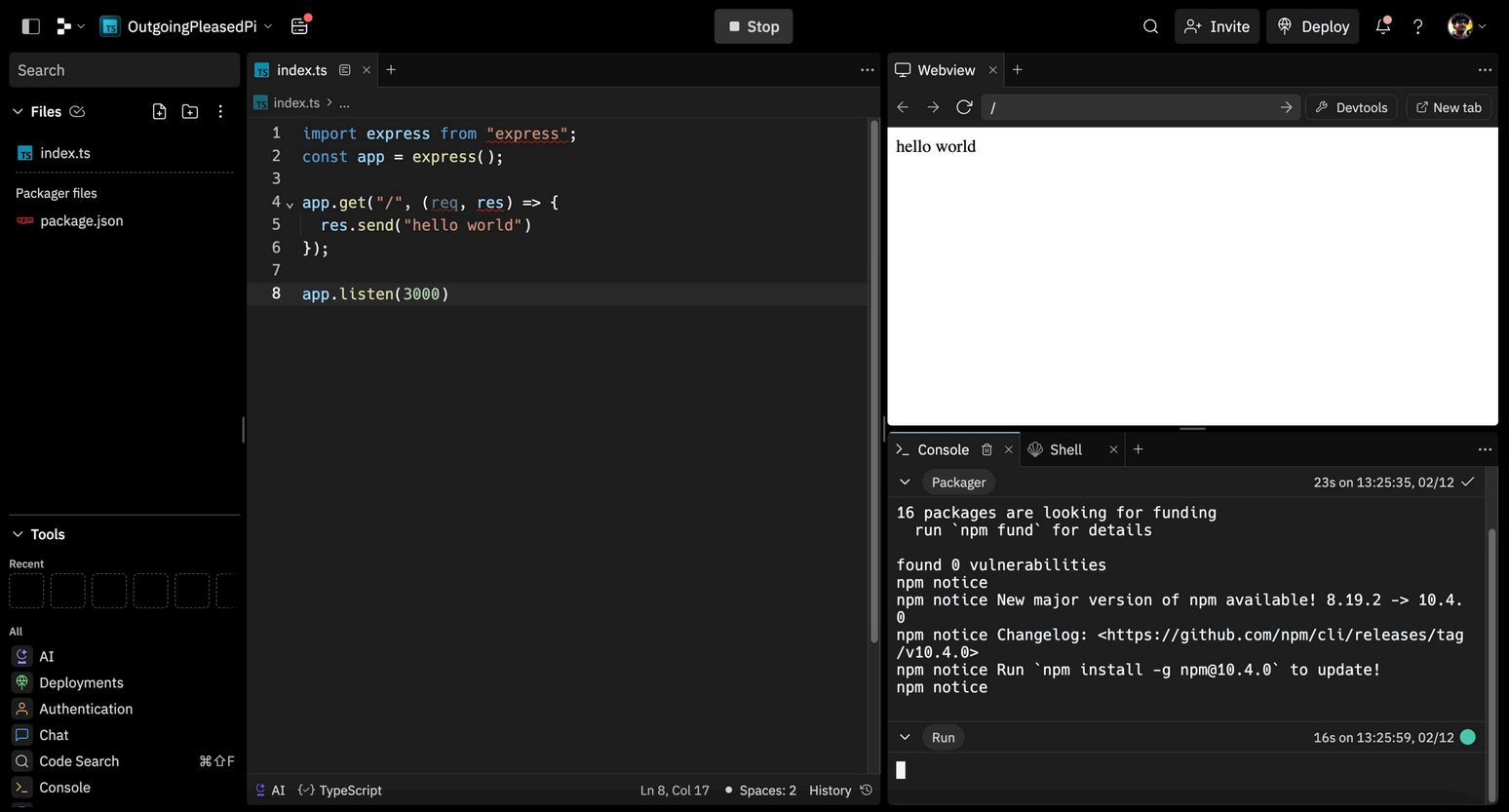
- **Codecademy**: Uses xterm.js in its courses on Bash.
- **Laravel Ssh Web Client**: Laravel server inventory with ssh web client to connect at server using xterm.js
- **Replit**: Collaborative browser based IDE with support for 50+ different languages.
- **TeleType**: cli tool that allows you to share your terminal online conveniently. Show off mad cli-fu, help a col
- **Intervue**: Pair programming for interviews. Multiple programming languages are supported, with results dis

What we’re building

Online IDE for long running backend/frontend apps

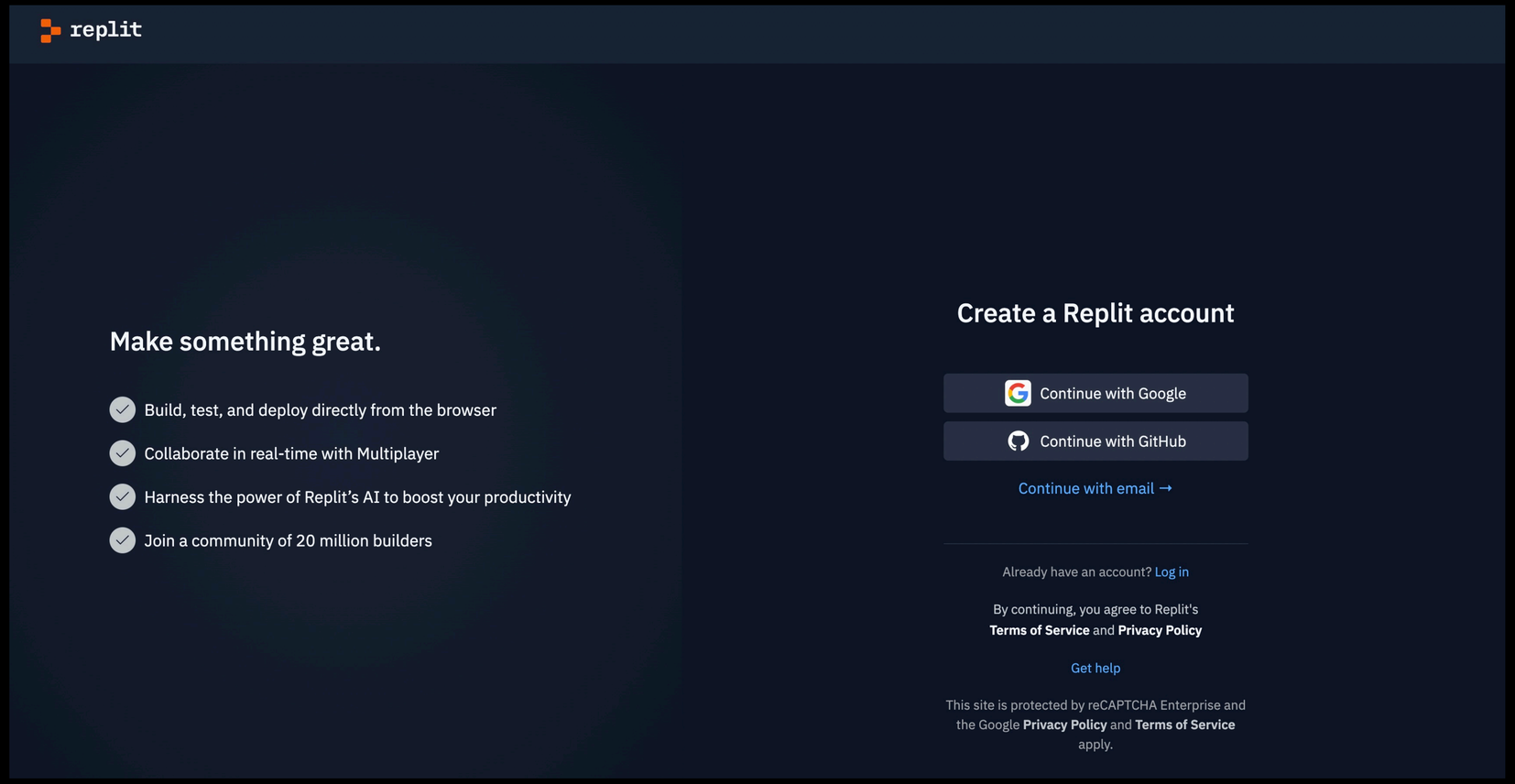


What we’re building



1. Online IDE for long running backend/frontend apps
2. Ability to start a fresh environment (in Rust, Go, Python, Node, React …)
3. Ability to autoscale servers with users
4. Run code in an isolated environment

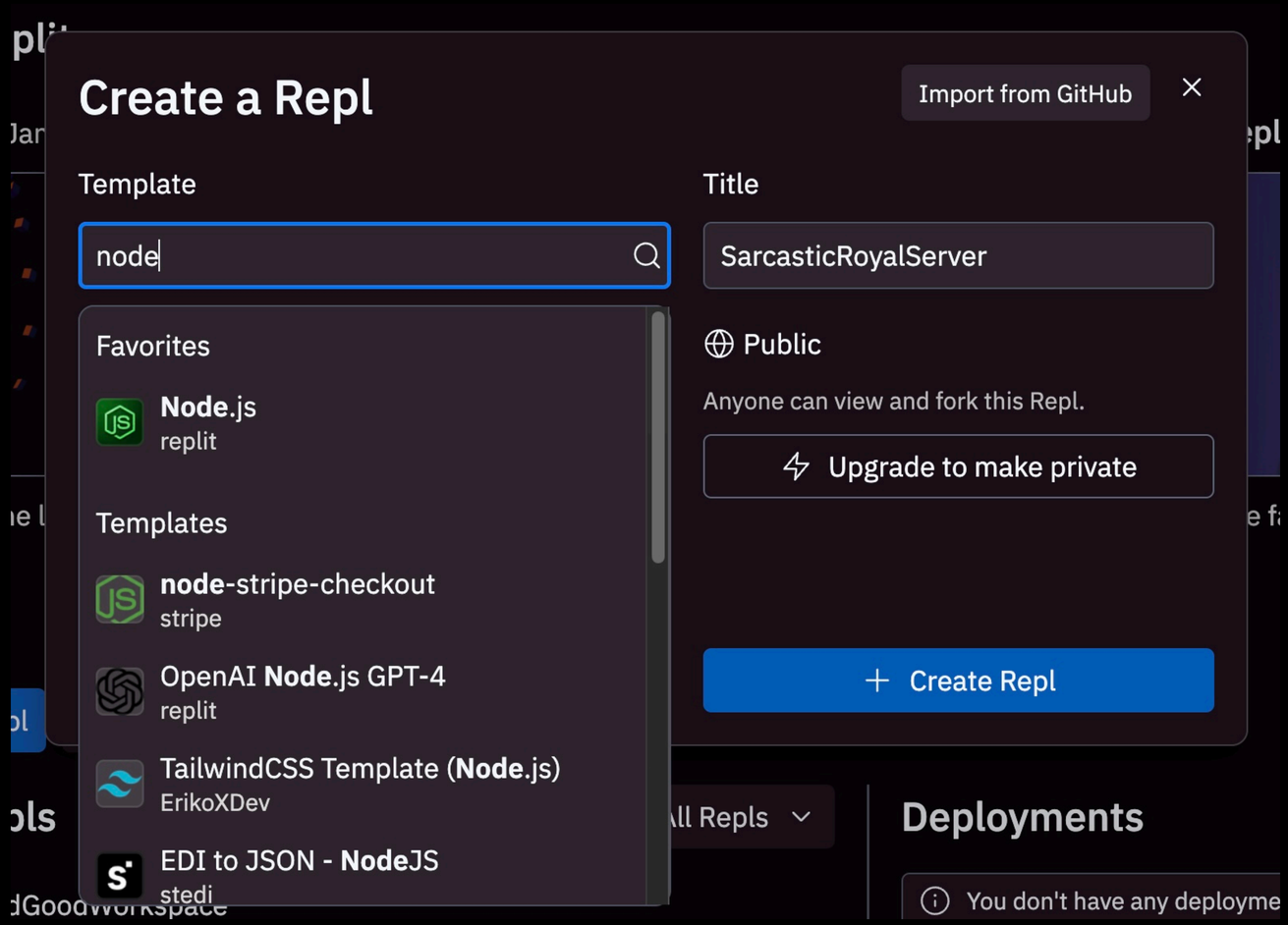
What we’re NOT building, but good to haves



1. **Authentication (Login with google)**
2. **Extremely good UI**

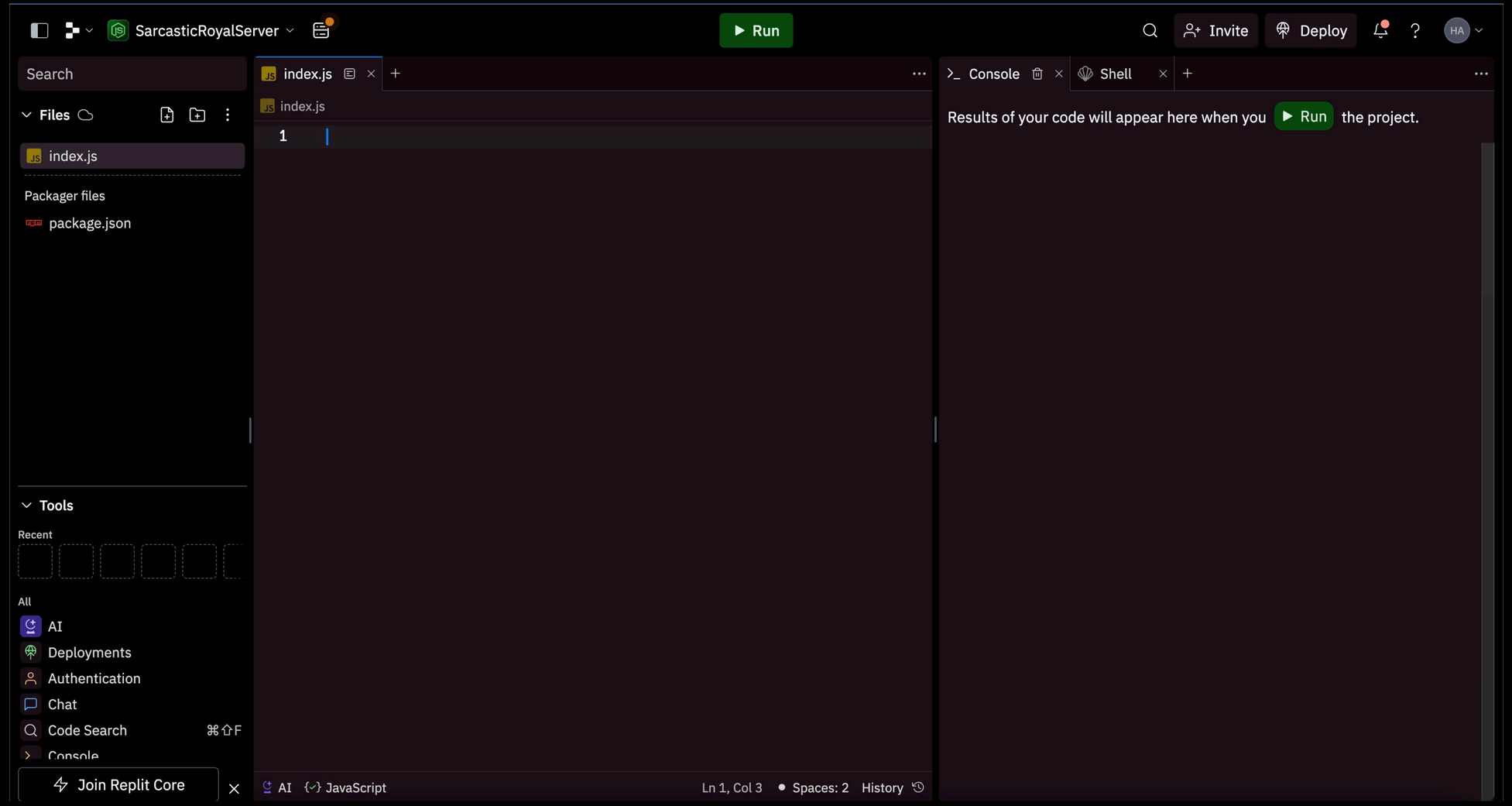
Requirements

User should be able to start a new environment in a selected stack



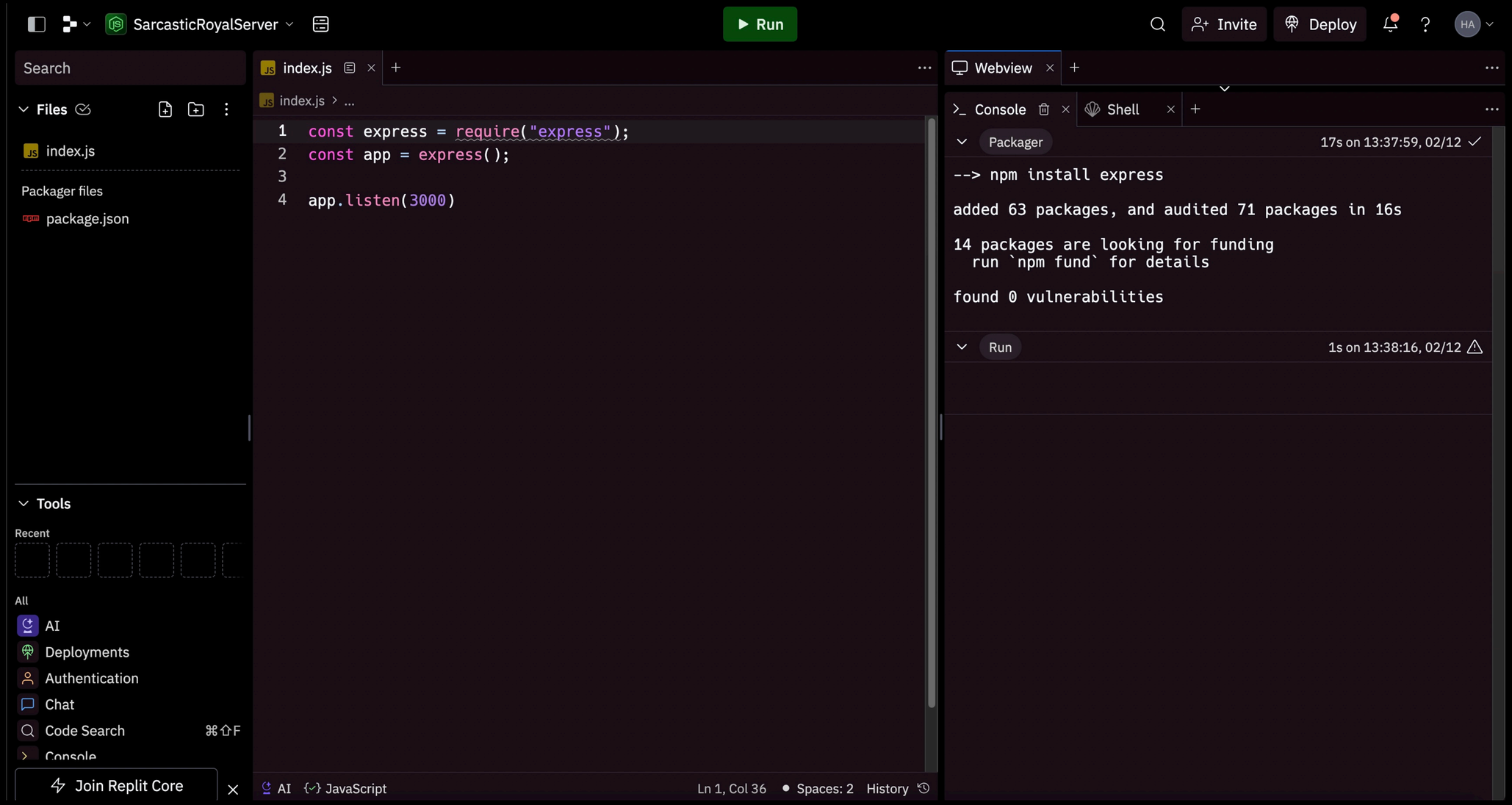
Requirements

User should be taken to a basic boilerplate repl for that environment



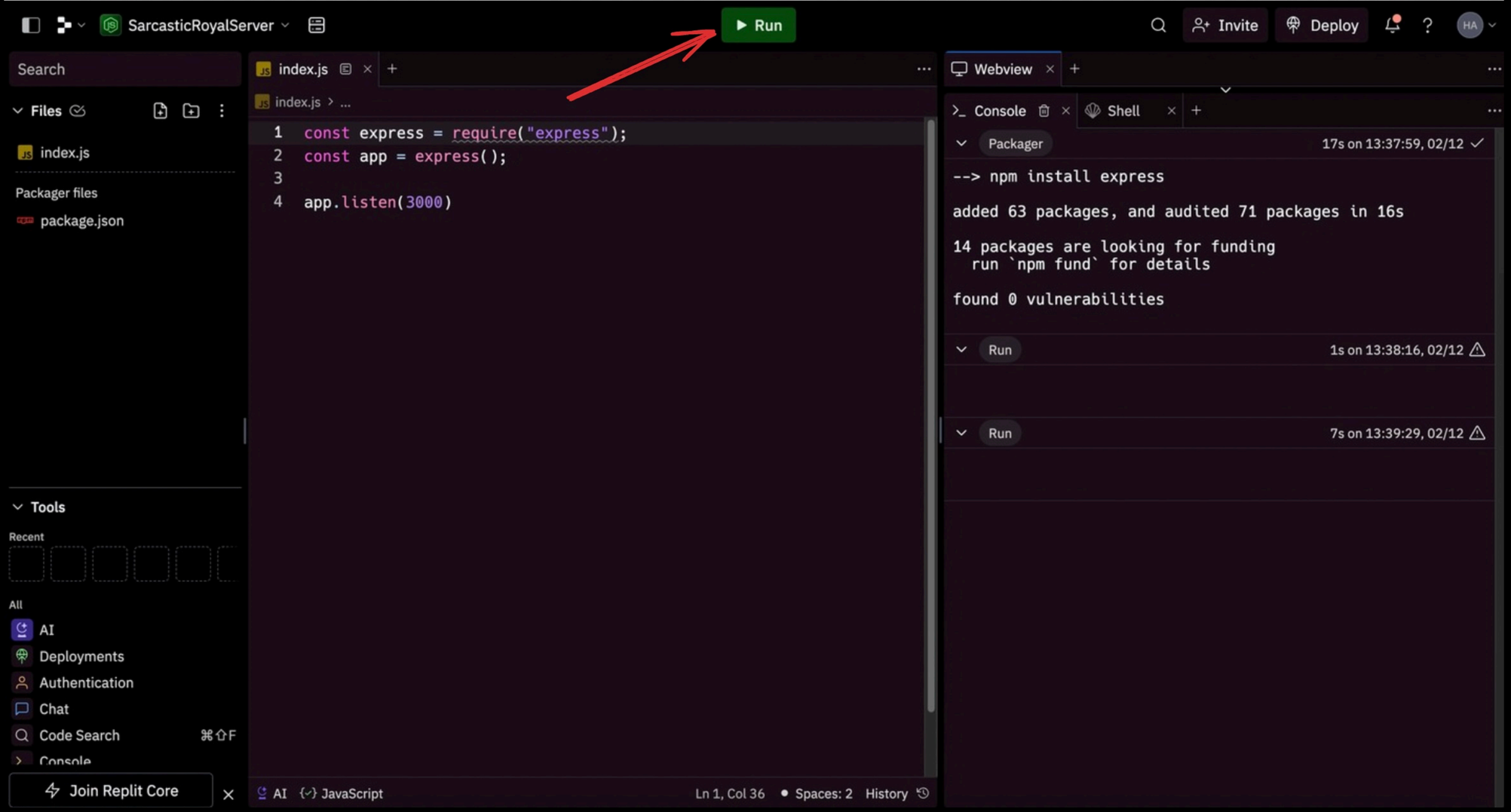
Requirements

User should be able to edit and save their code somewhere



Requirements

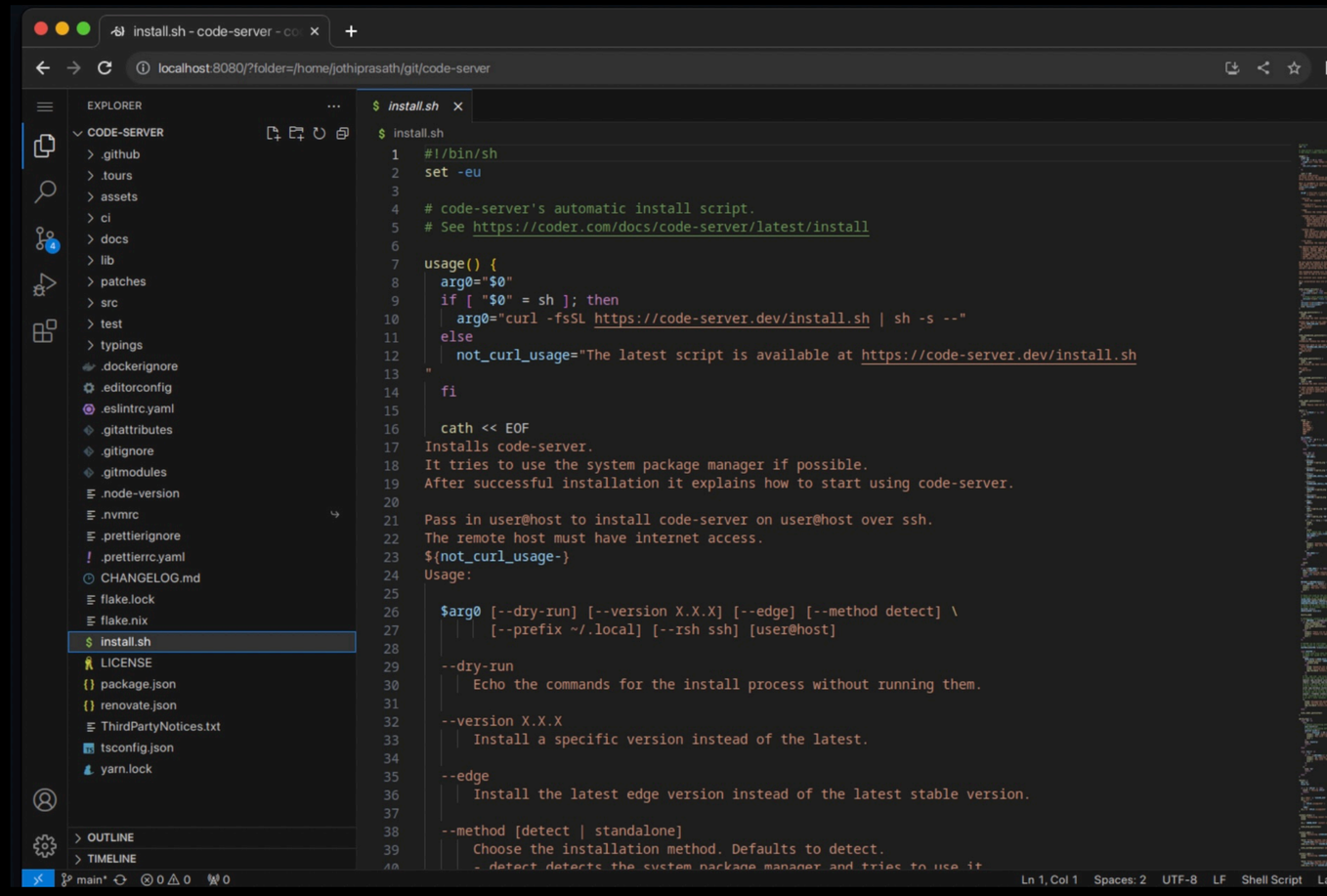
User should be able to run code (both long running and short running)



We'll be taking 3 approaches to solve this problem

Callout -
You can use an external service/codebase that does a lot of this for you.
If you're a startup, you probably want to pick one of these services and not build this from scratch

https://github.com/coder/code-server
(Github codespaces)



Beginner friendly

Great way to build intuition on how you can build something like this
DO NOT use it in production/in an interview

Why you shouldn't do it

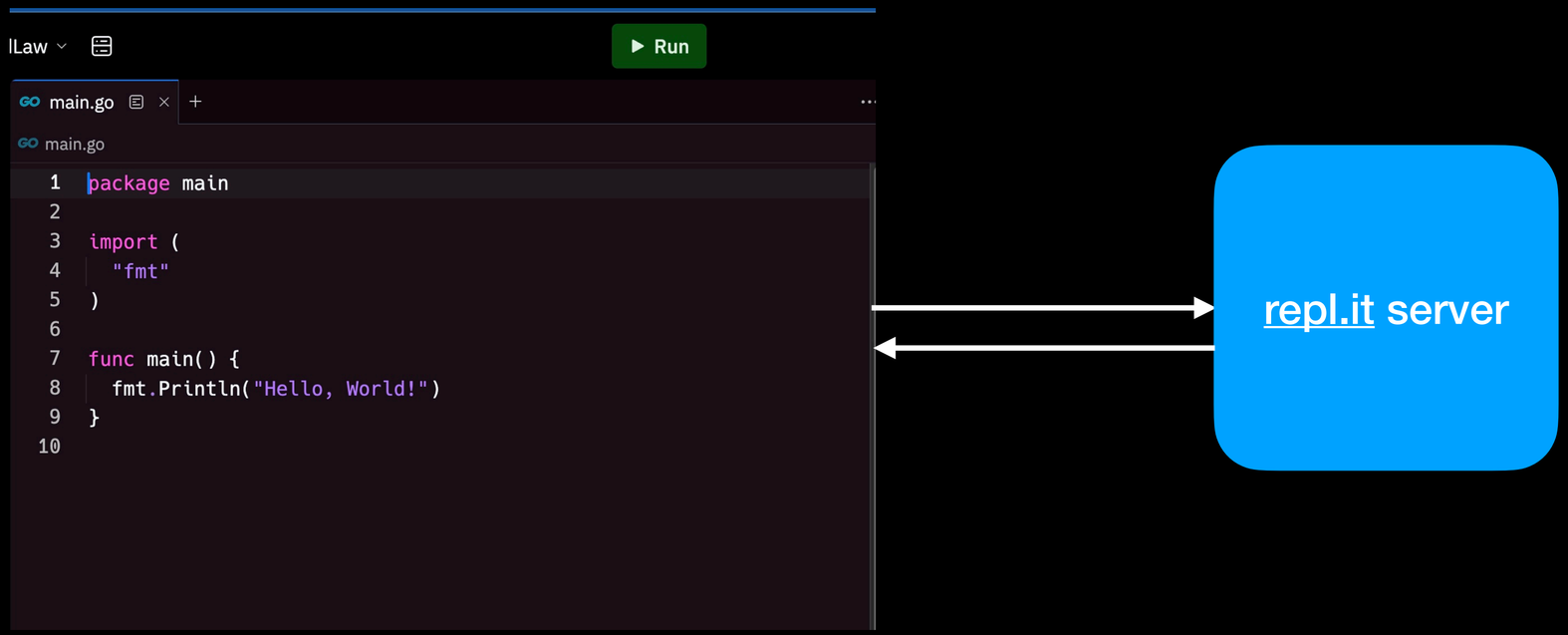
Downsides

1. Insecure Remote code execution
2. Single server setup, doesn't autoscale
3. Port conflicts between two users (every user is sharing resources on the same server)
4. Terminal is extremely rudimentary
5. Very ugly package management

Why is building repl.it hard?

Why is building repl.it hard?

1. Remote code execution

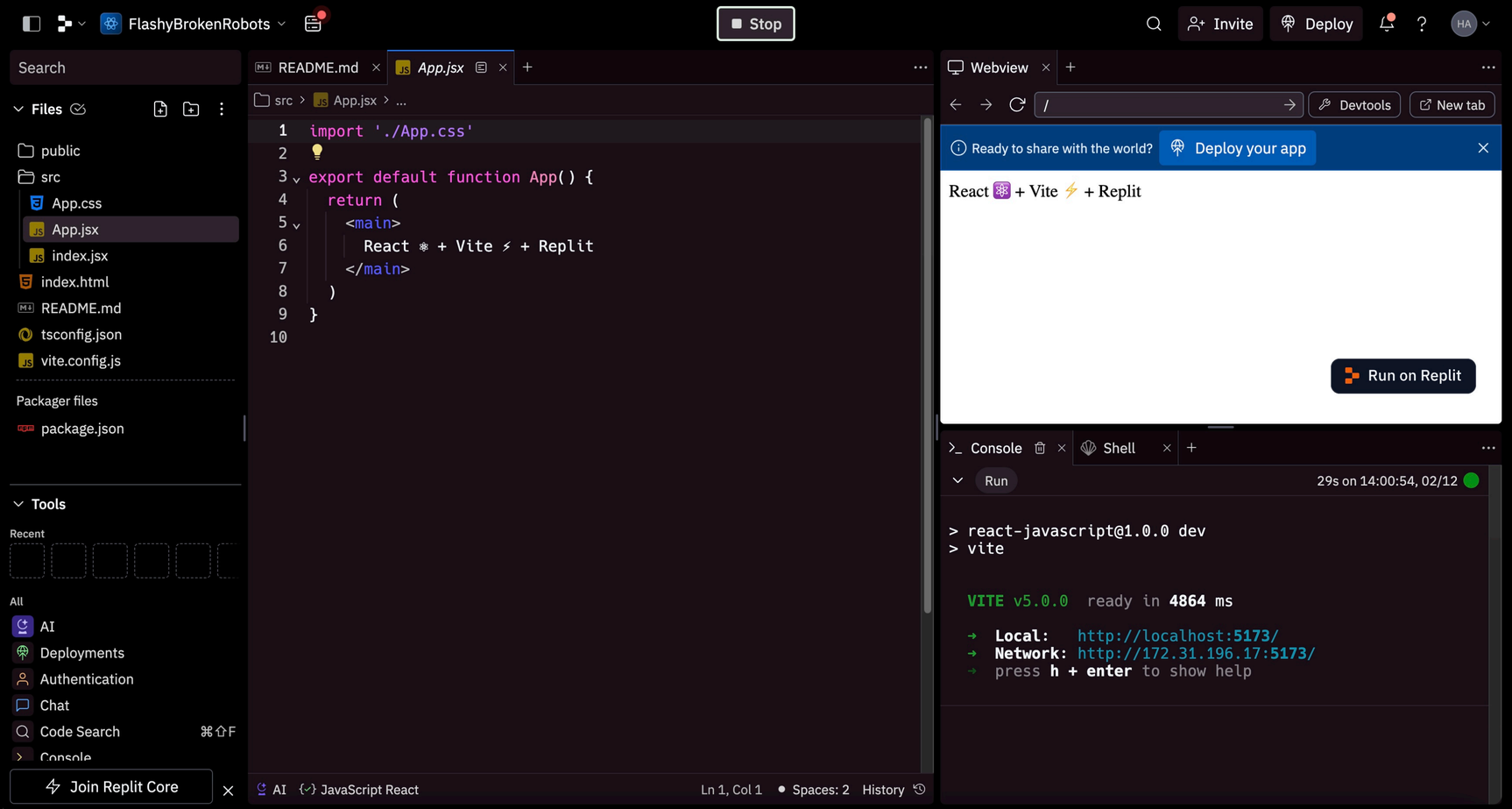


Your browser can’t run C++/Rust/Go/React (Not technically true)

Why is building repl.it hard?

1. Remote code execution
2. Long running processes

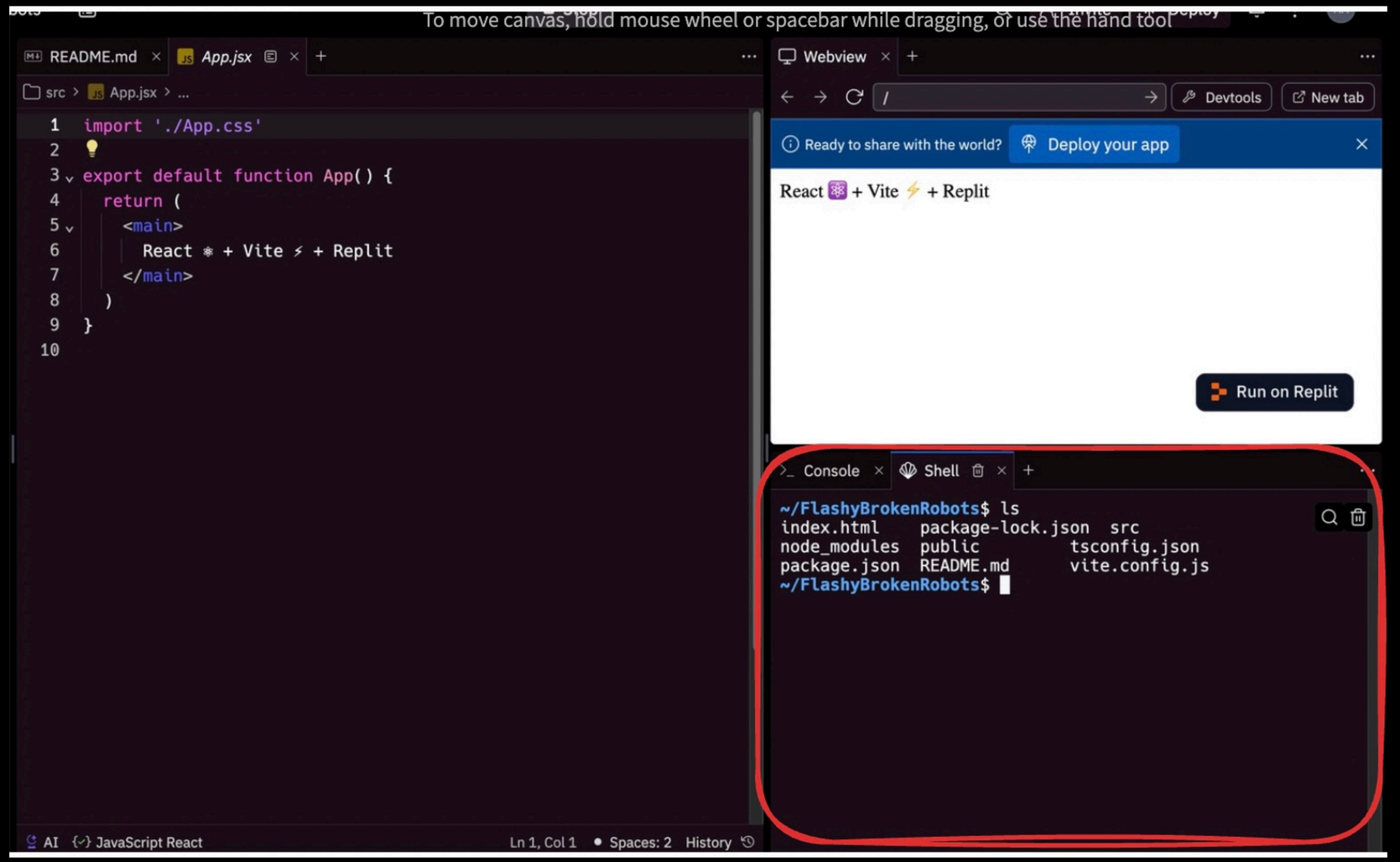
You need to give user VCPU guarantees



Example of short running process - Leetcode

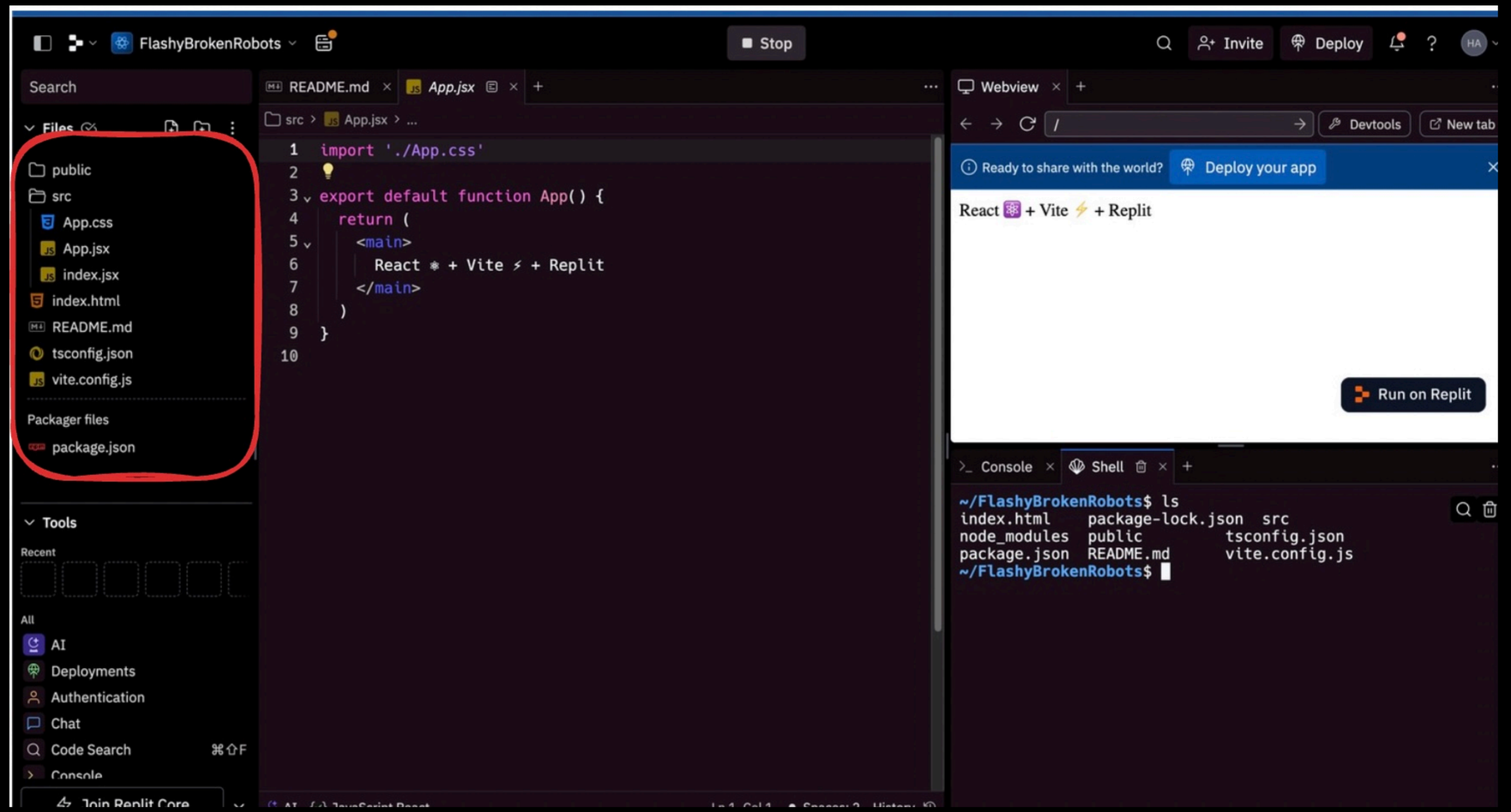
Why is building repl.it hard?

1. Remote code execution
2. Long running processes
3. Shell access inside browser



Why is building repl.it hard?

1. Remote code execution
2. Long running processes
3. Shell access inside browser
4. File storage (not difficult)



Example of not requiring file storage - Leetcode

Why is building repl.it hard?

Let’s see how we can achieve all of these one by one

1. Remote code execution
2. Long running processes
3. Shell access inside browser
4. File storage (not difficult)

You will have 1 big monolith for this part

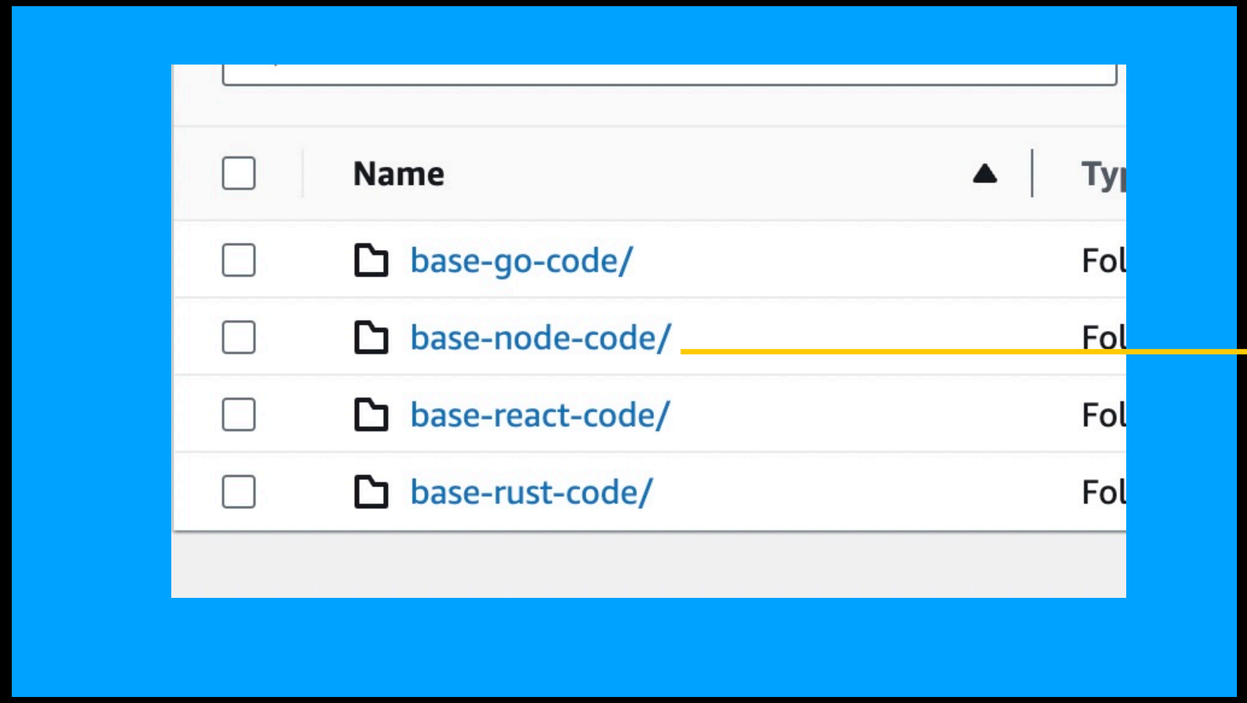
Execution service

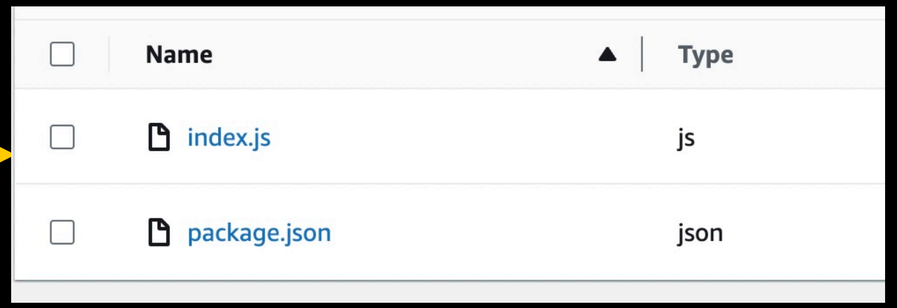
1. Remote code execution
2. Long running processes
3. Shell access inside browser
4. File storage (not difficult)

Execution service

Step 0.1 - Keeping a copy of base projects in S3

AWS S3

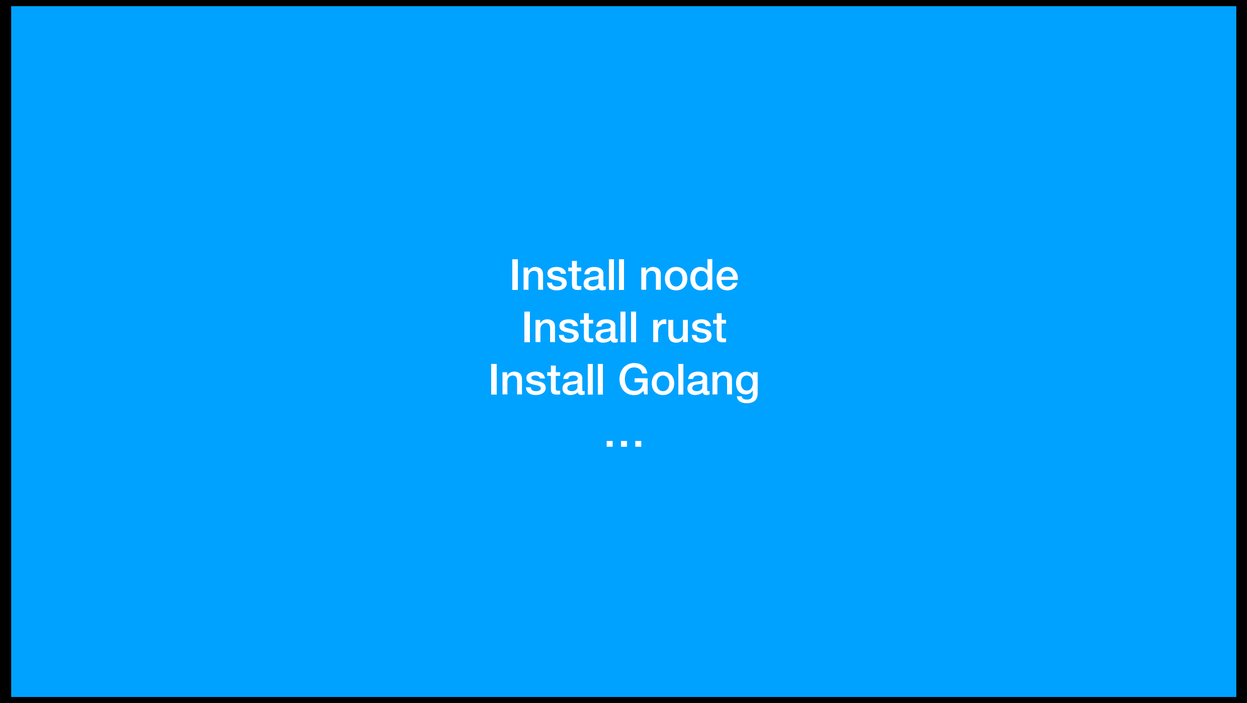




Execution service

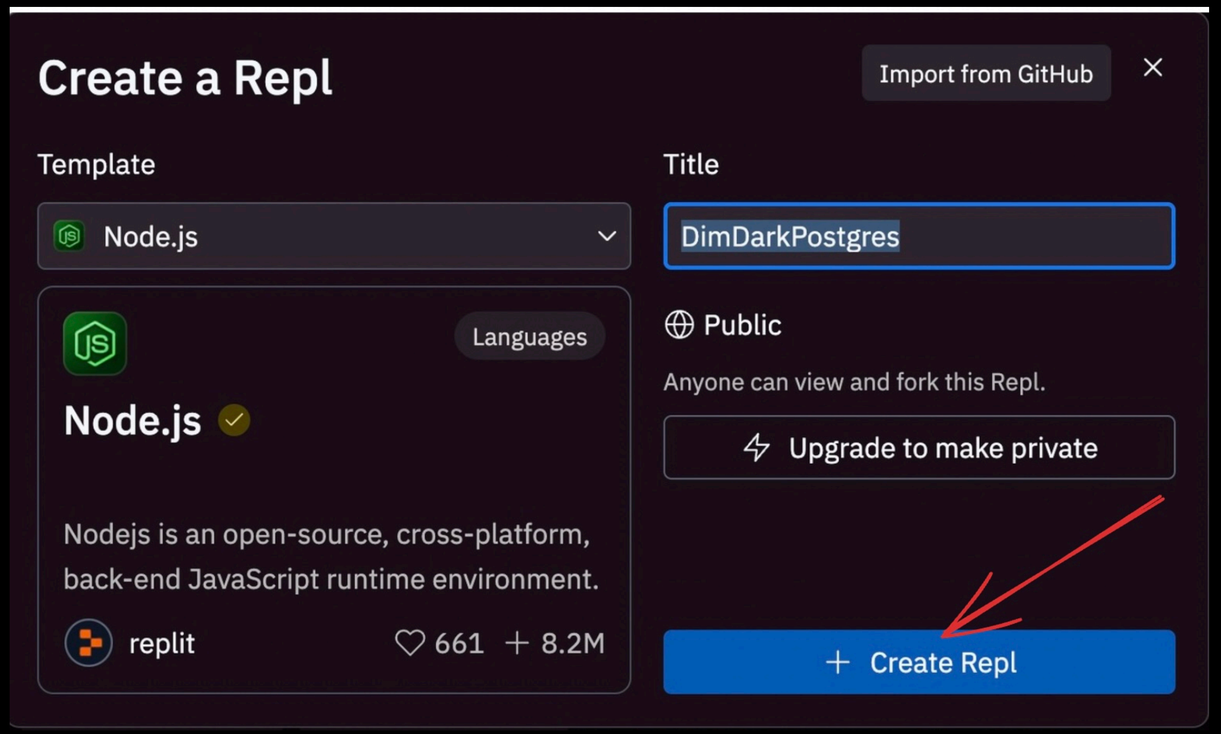
Step 0.2 - Bringing all languages you support (Node, Go, Rust) to this machine

Mac machine/AWS Server (Execute service)



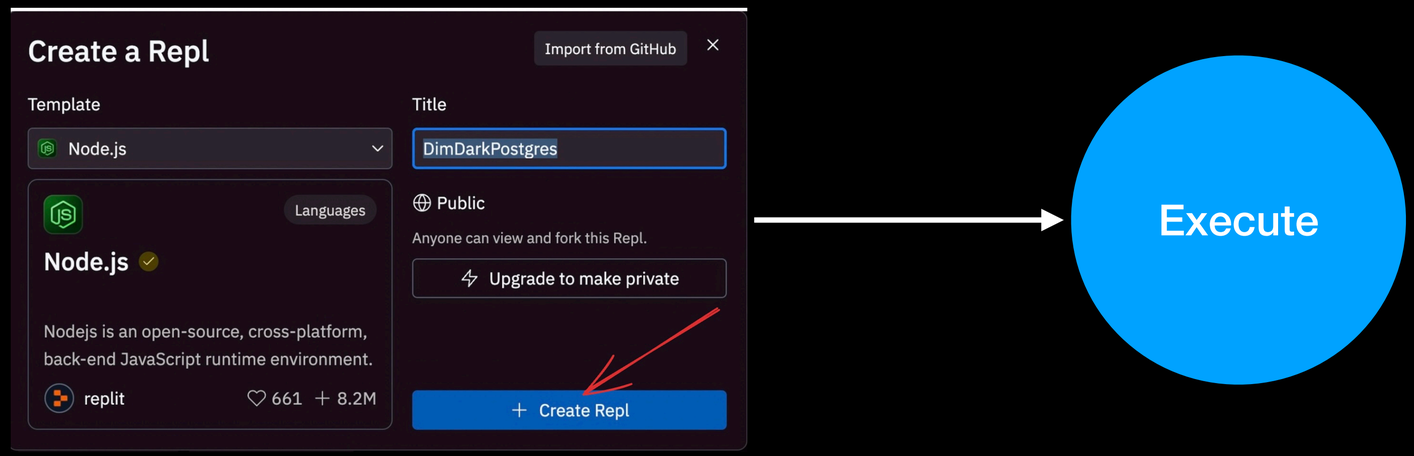
Execution service

Step 1 - Initialising the repl



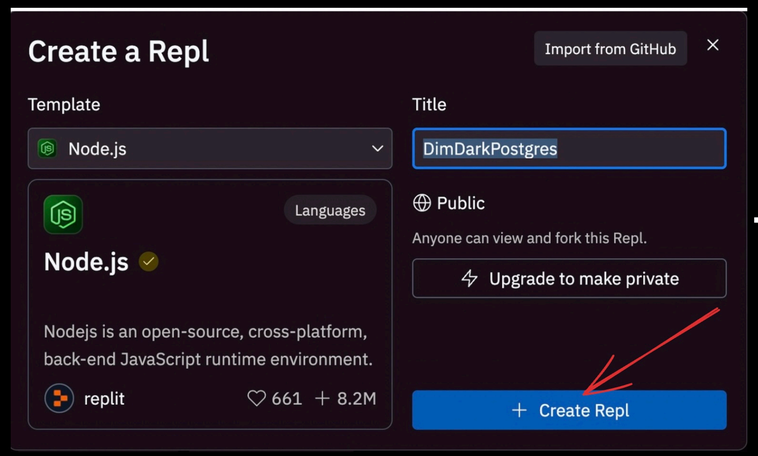
Execution service

Step 1 - Initialising the repl



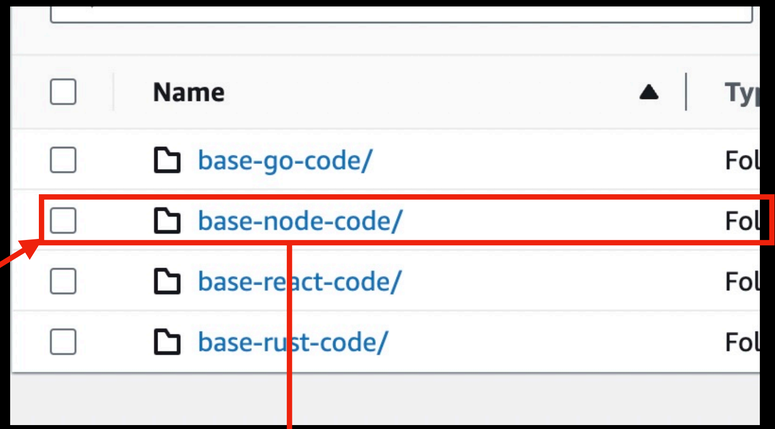
Execution service

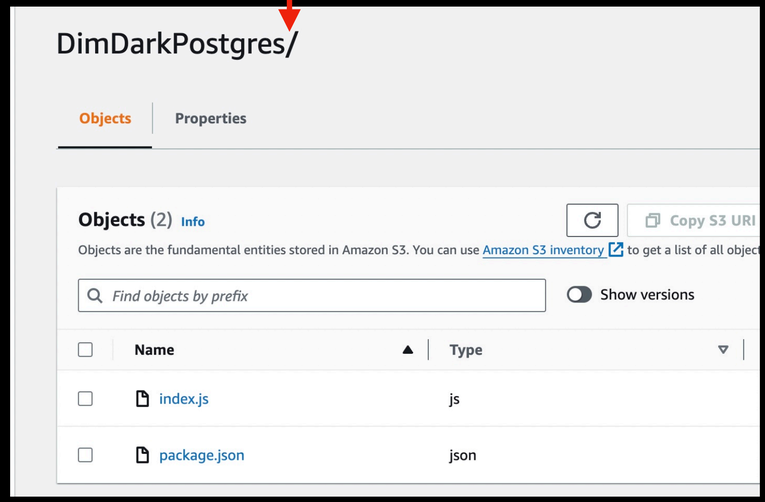
Step 1 - Initialising the repl



Execute

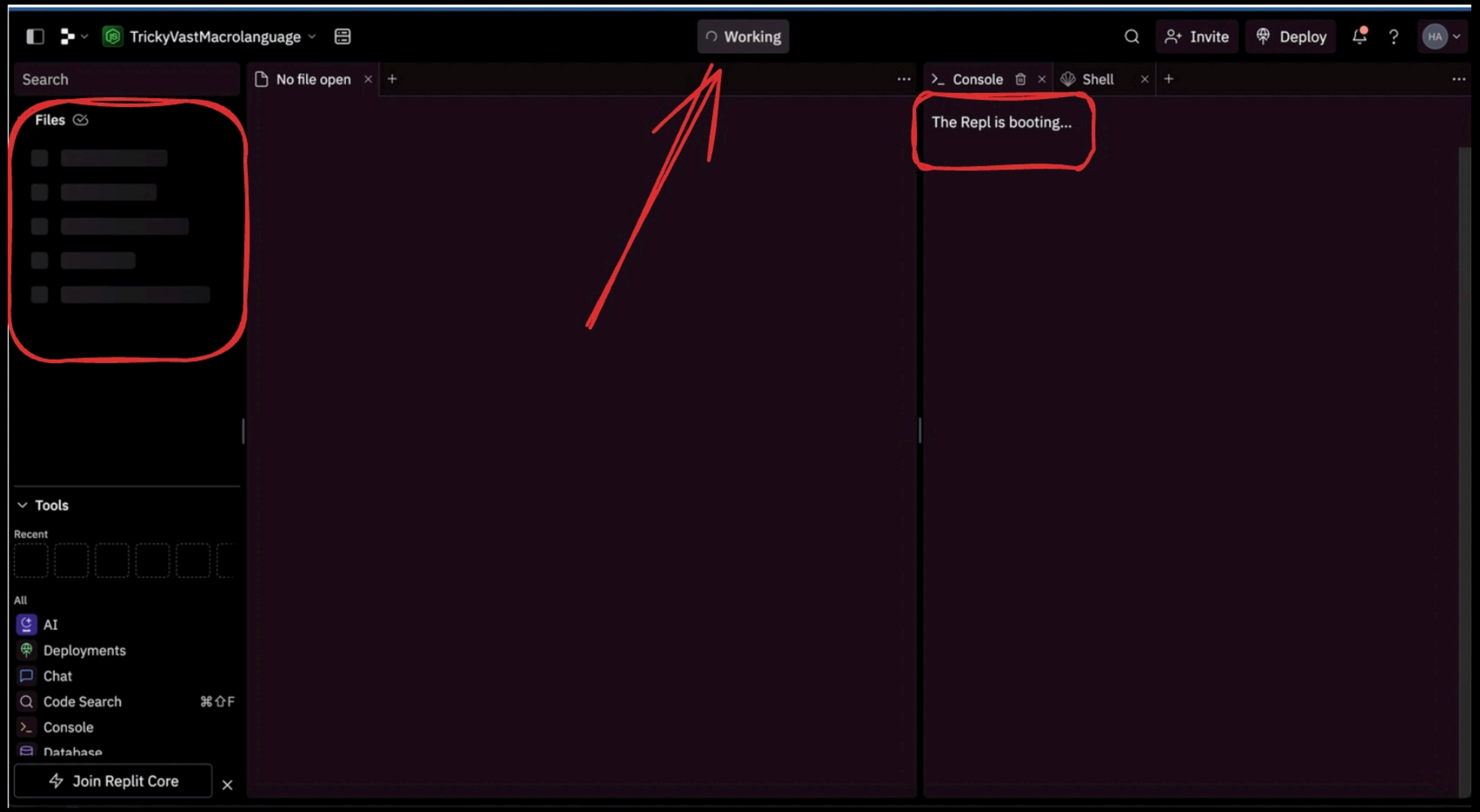
Copy over the base image to s3://images/{id}





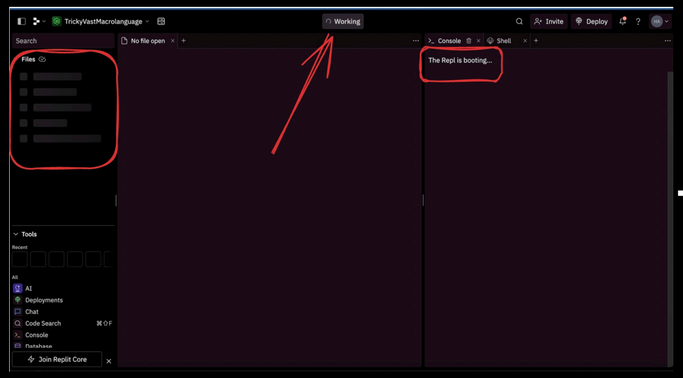
Execution service

Step 2 - Taking the user to the edit screen



Execution service

Step 3 - Initialise a ws connection

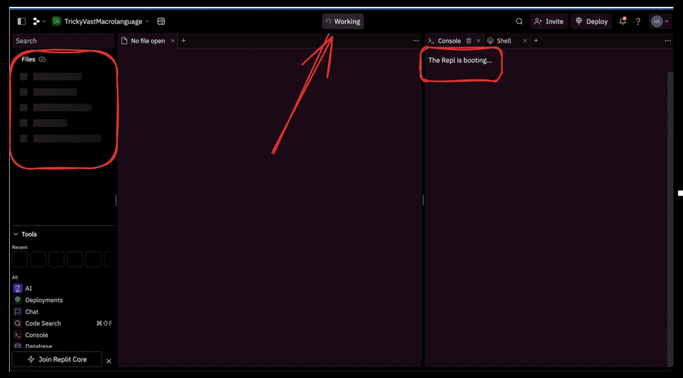


Websocket

Execute

Execution service

Step 4 -Bring the users code to the VM



Websocket

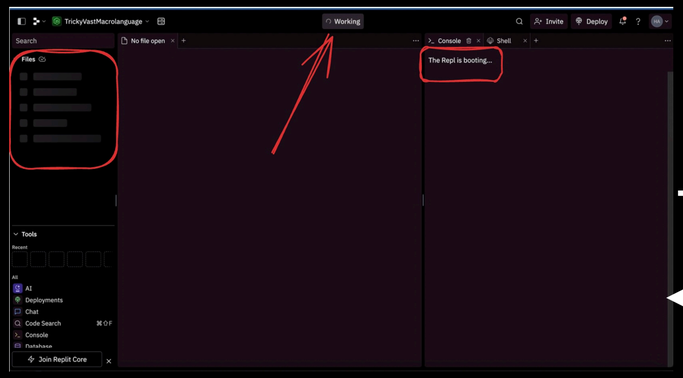
Execute

S3

Pulls latest code

Execution service

Step 4 -Bring the users code to the VM



Websocket

Send the filesystem over to the user lazily

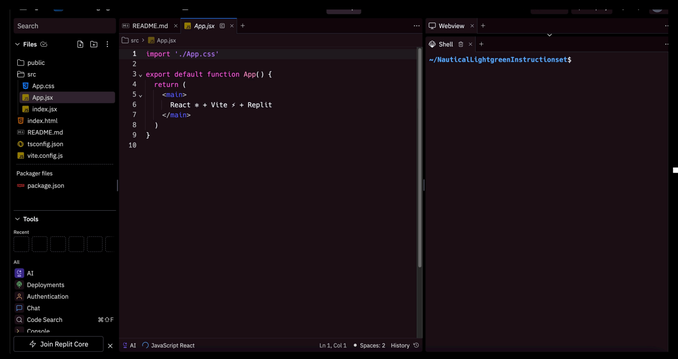
Execute

S3

Pulls latest code

Execution service

Step 5 - Let the user edit files



Websocket

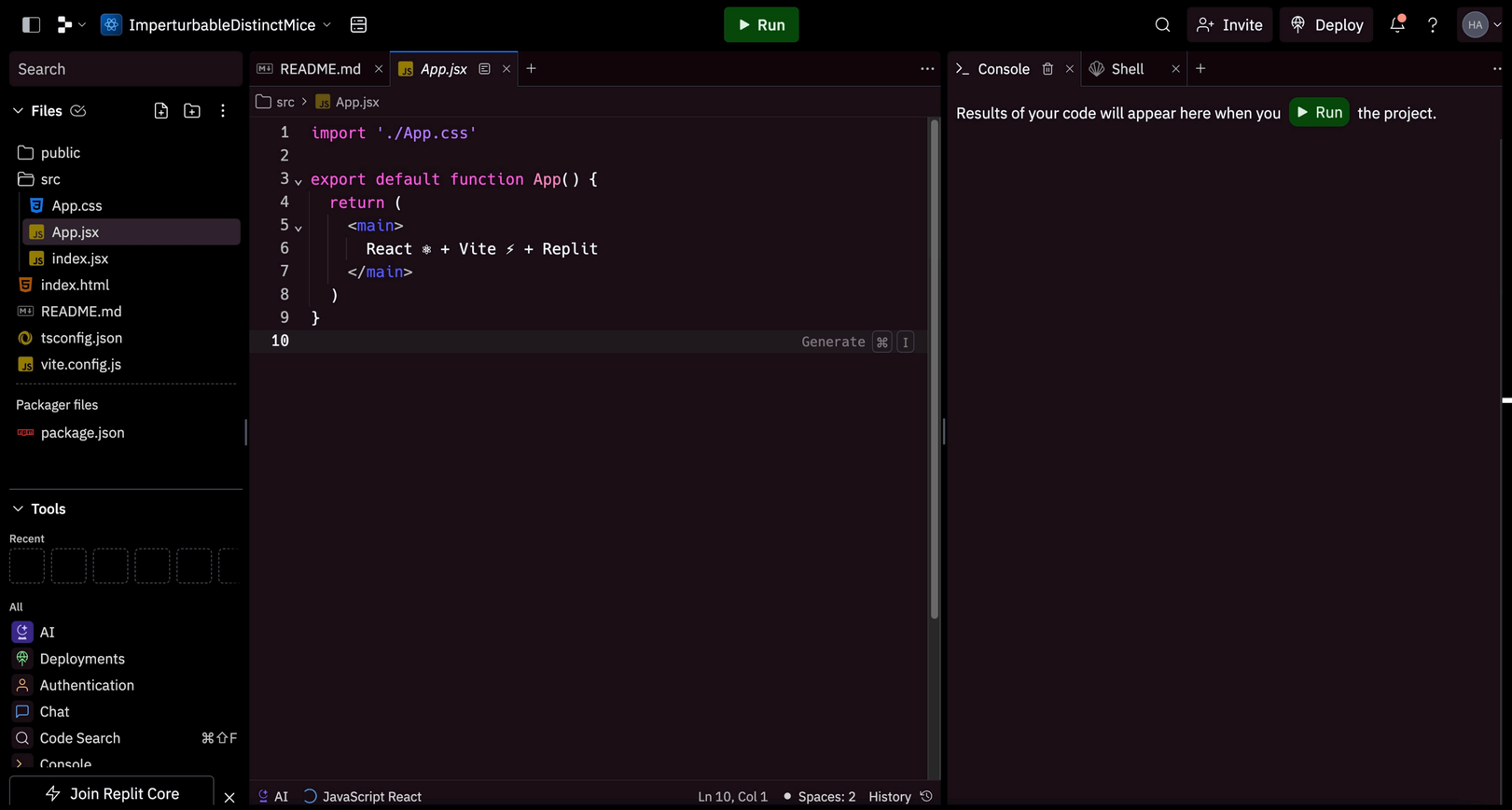
Execute

Pulls latest code

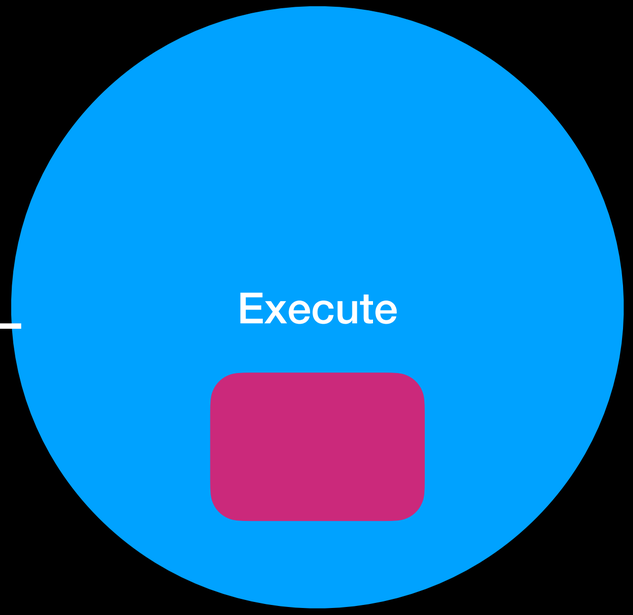
S3

Execution service

Step 5 - Let the user edit files

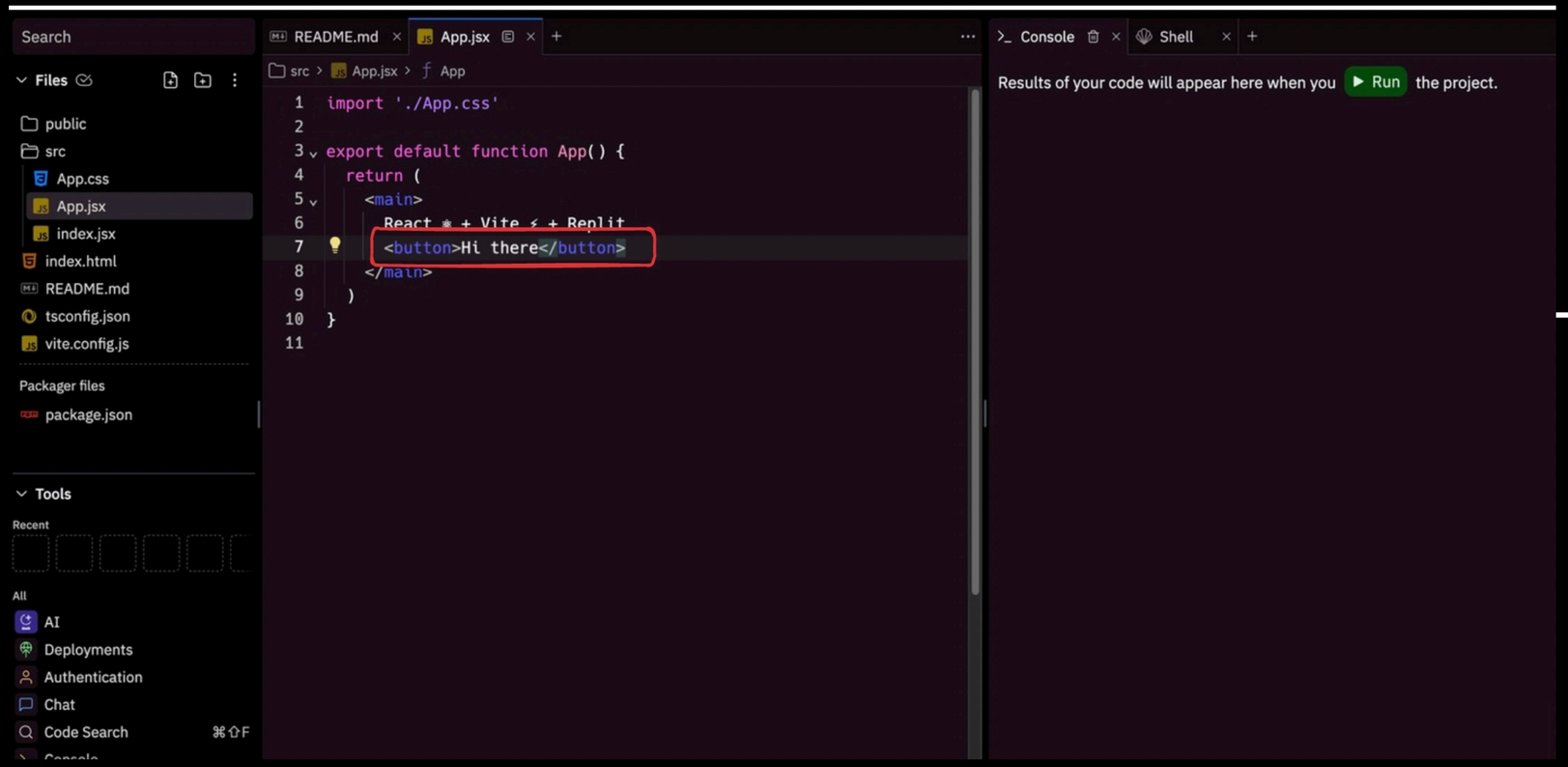


Websocket

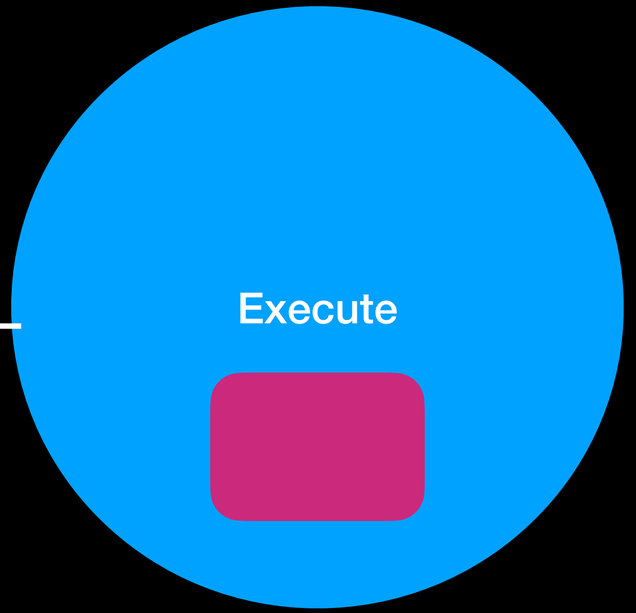


Execution service

Step 5 - Let the user edit files

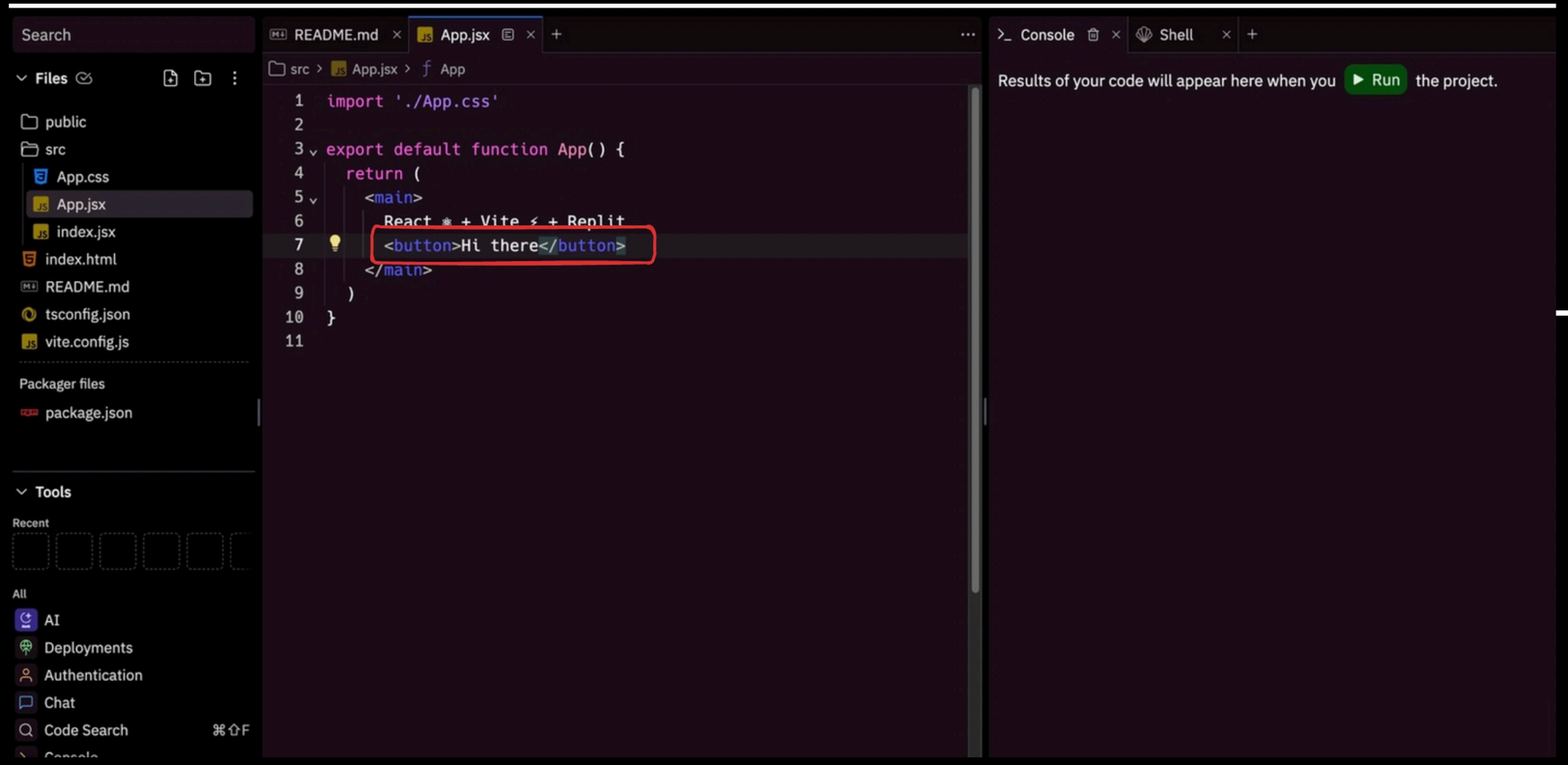


Websocket

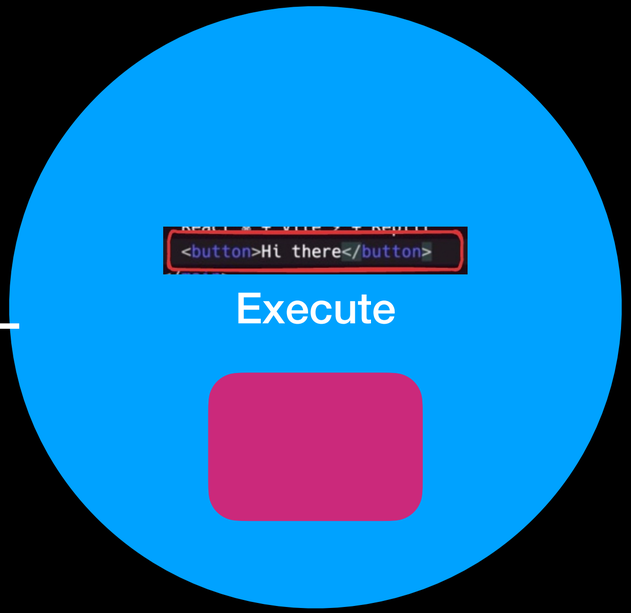


Execution service

Step 5 - Let the user edit files

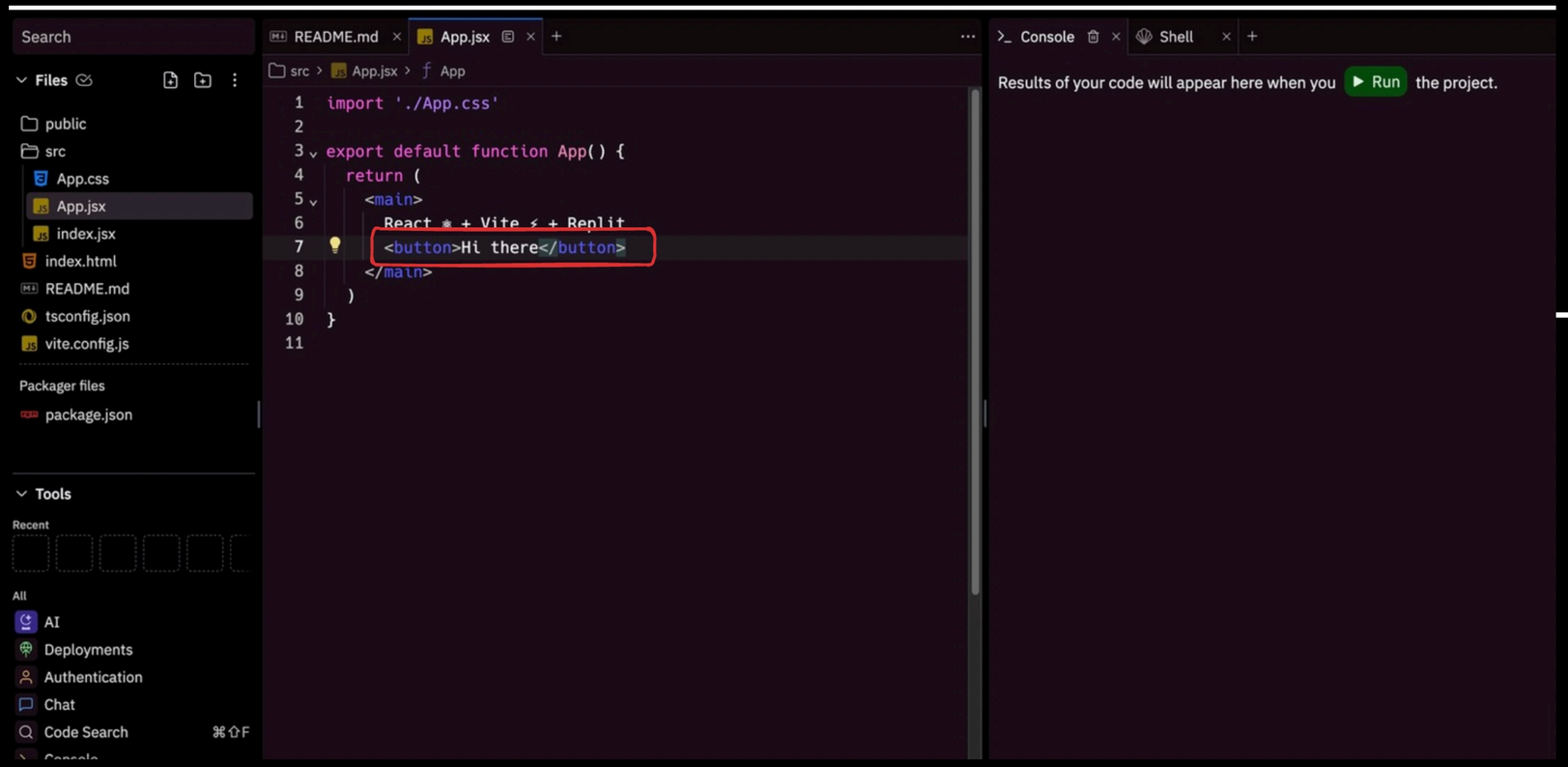


Websocket

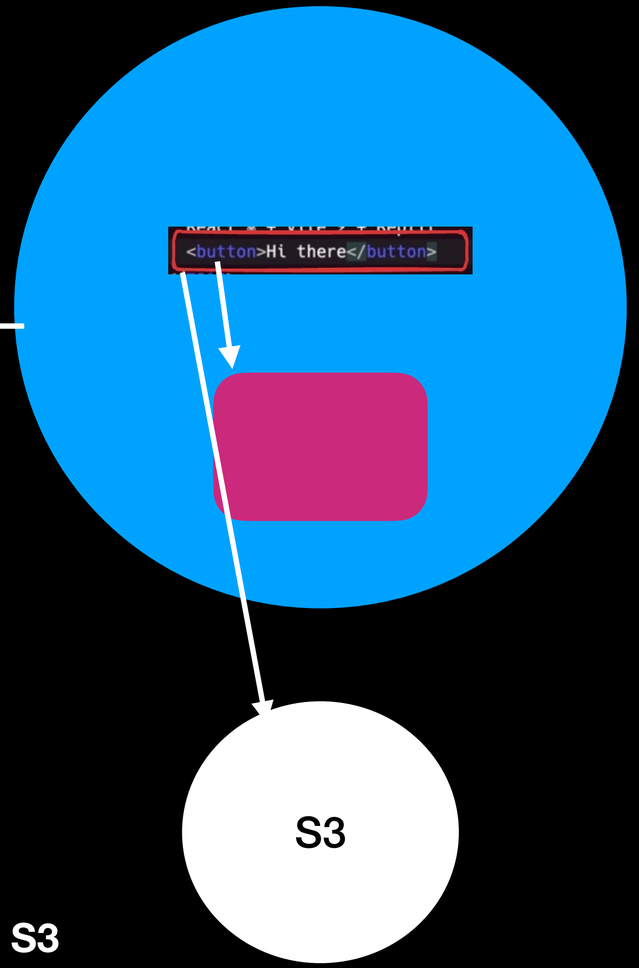


Execution service

Step 5 - Let the user edit files



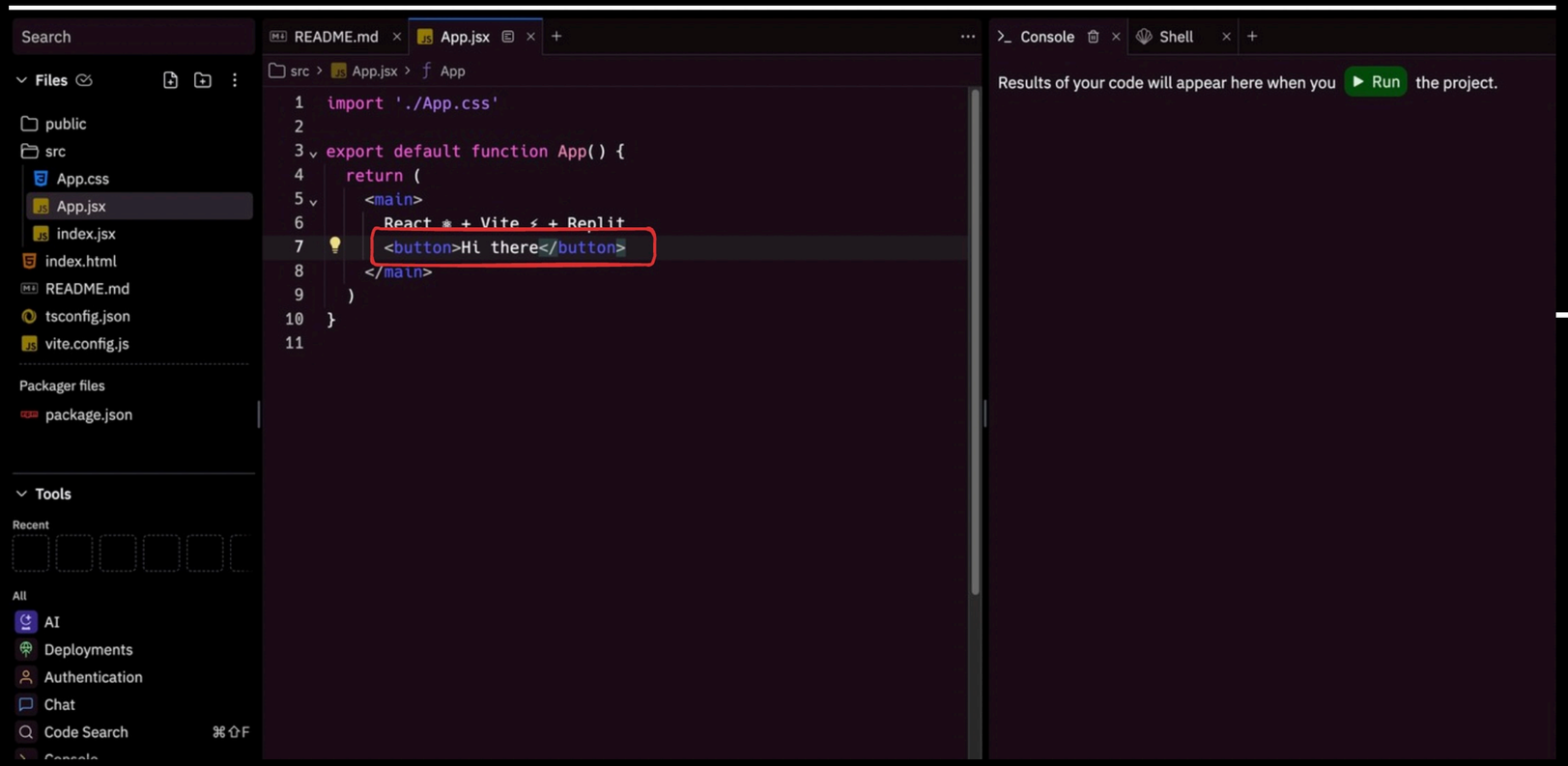
Websocket



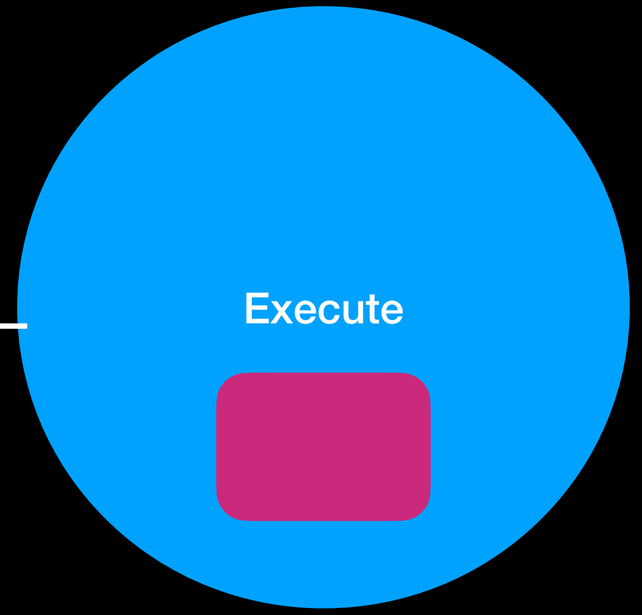
Callouts - Debounce these saves
You can mount a directory to S3 as well, although need to make sure node_modules don't reach S3

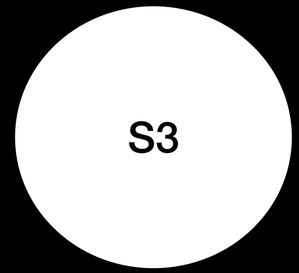
Execution service

Logic to add and delete files also remains the same!
Validation of files (file format, size) is something you should take into consideration



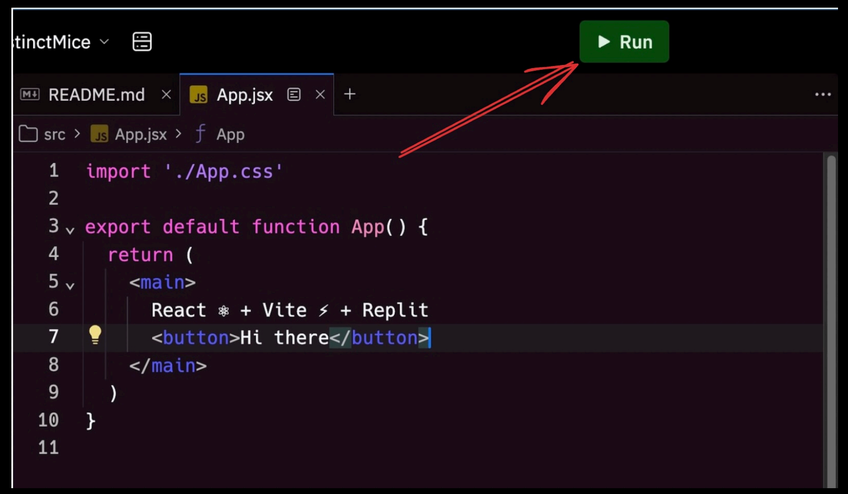
Websocket

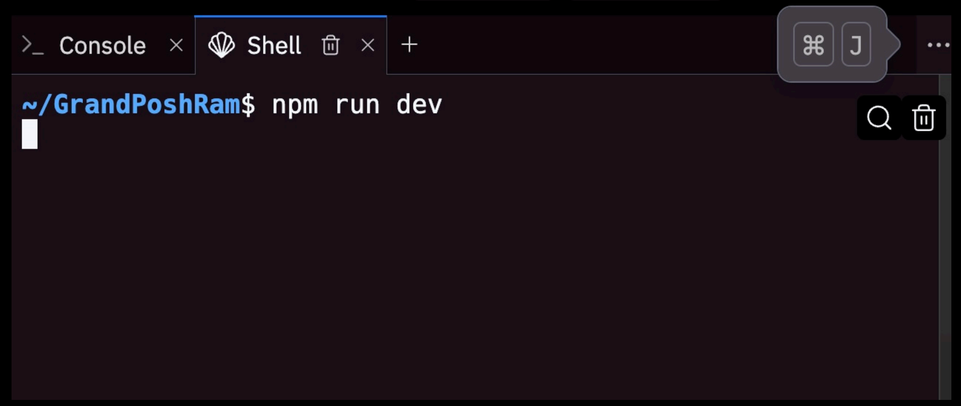


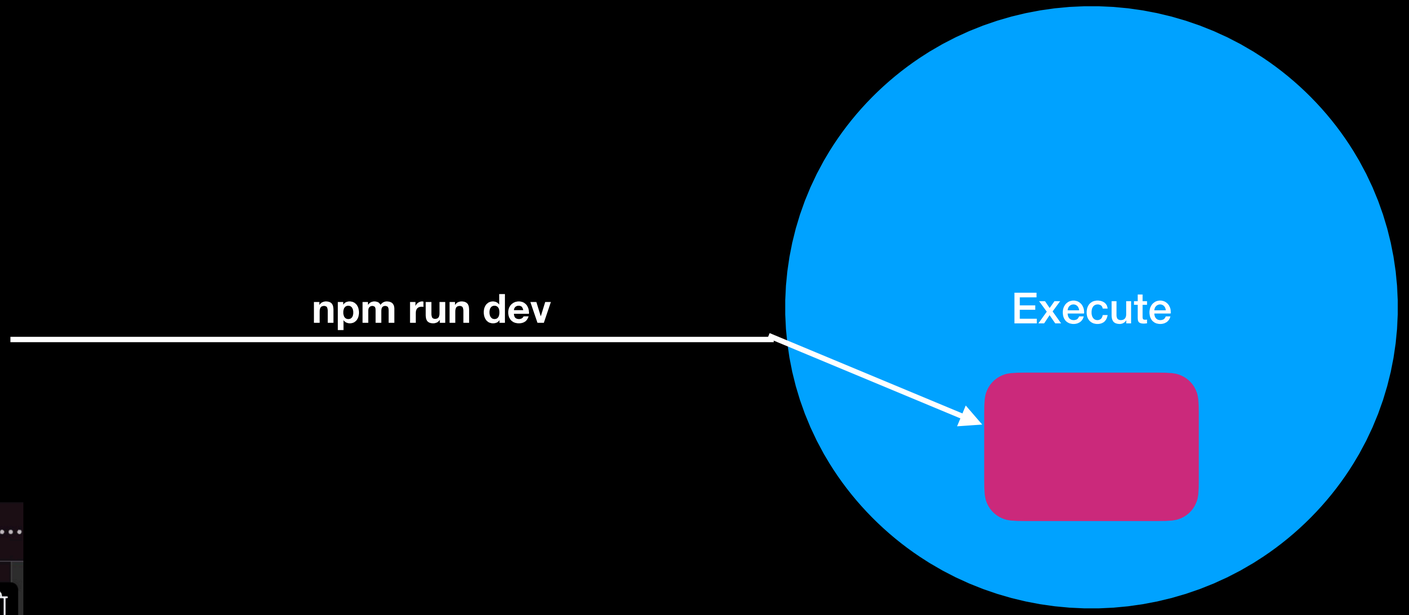


Execution service

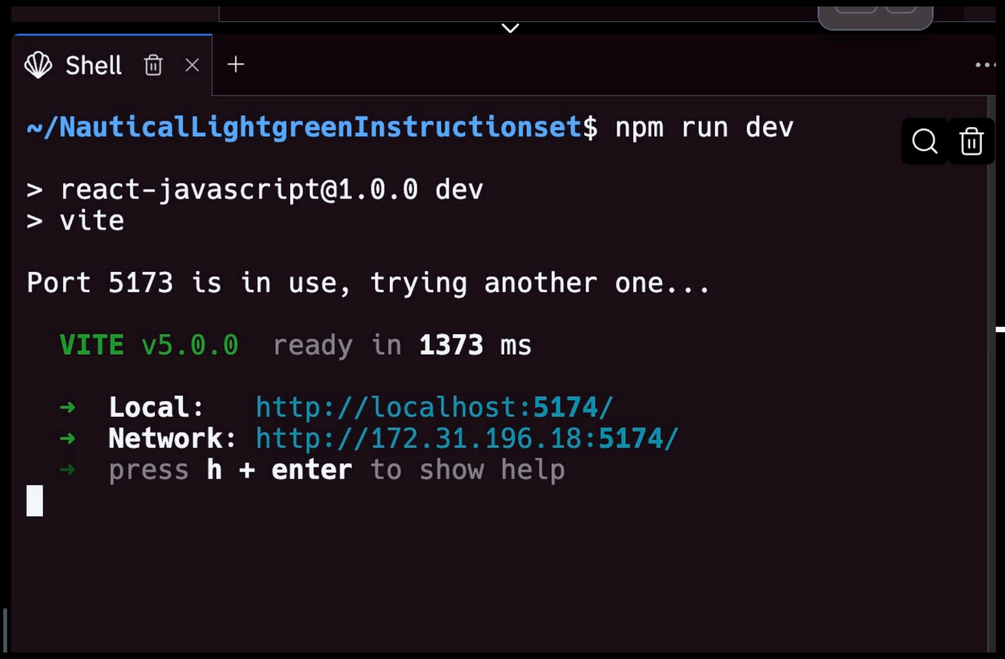
Step 6 -Running/Executing the code





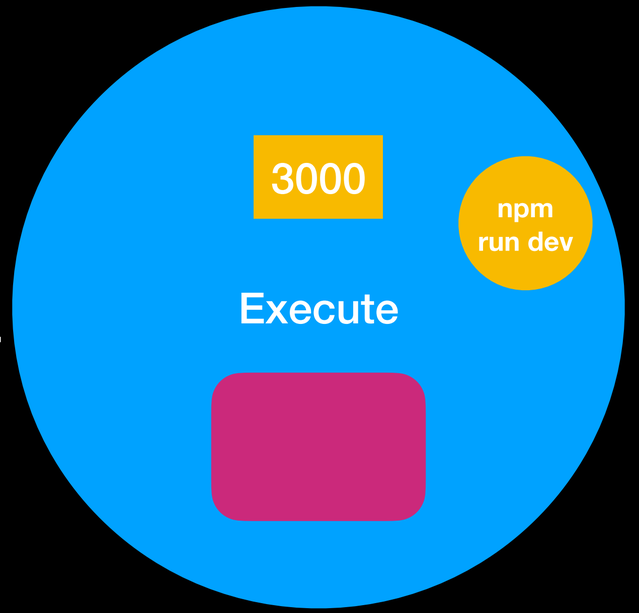


Execution service

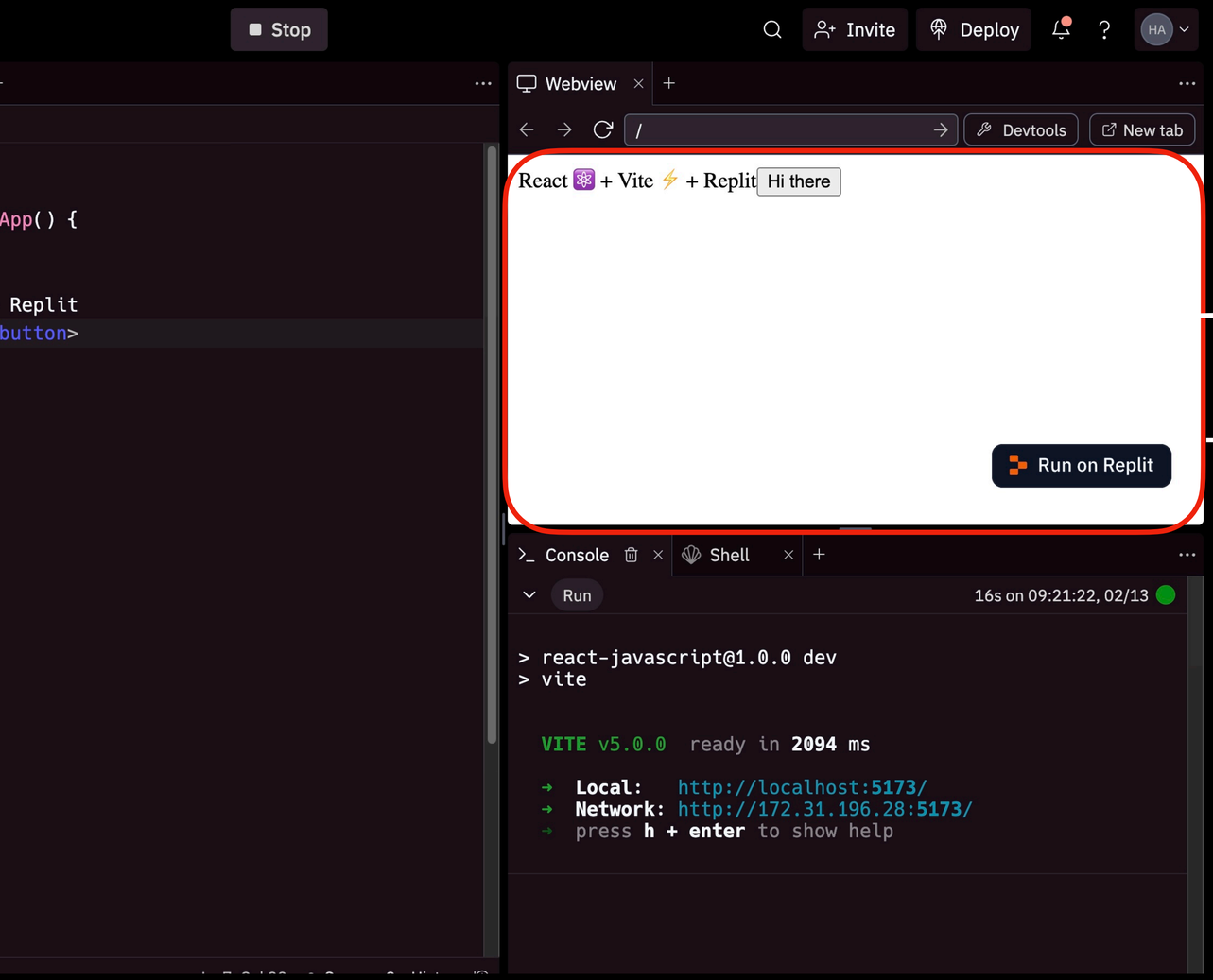


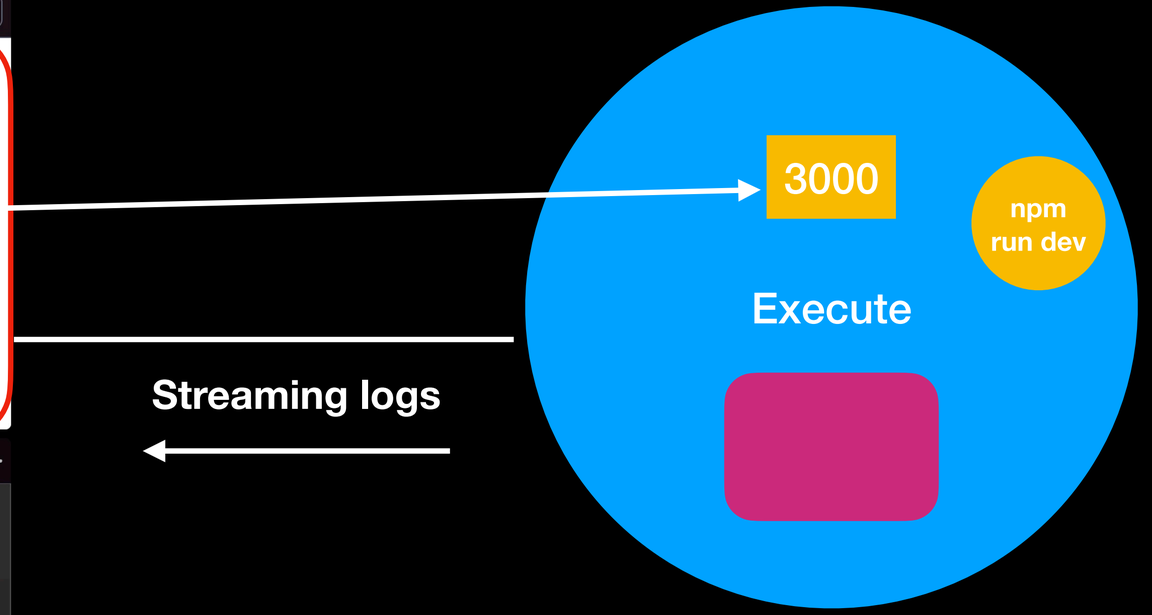
npm run dev

Streaming logs



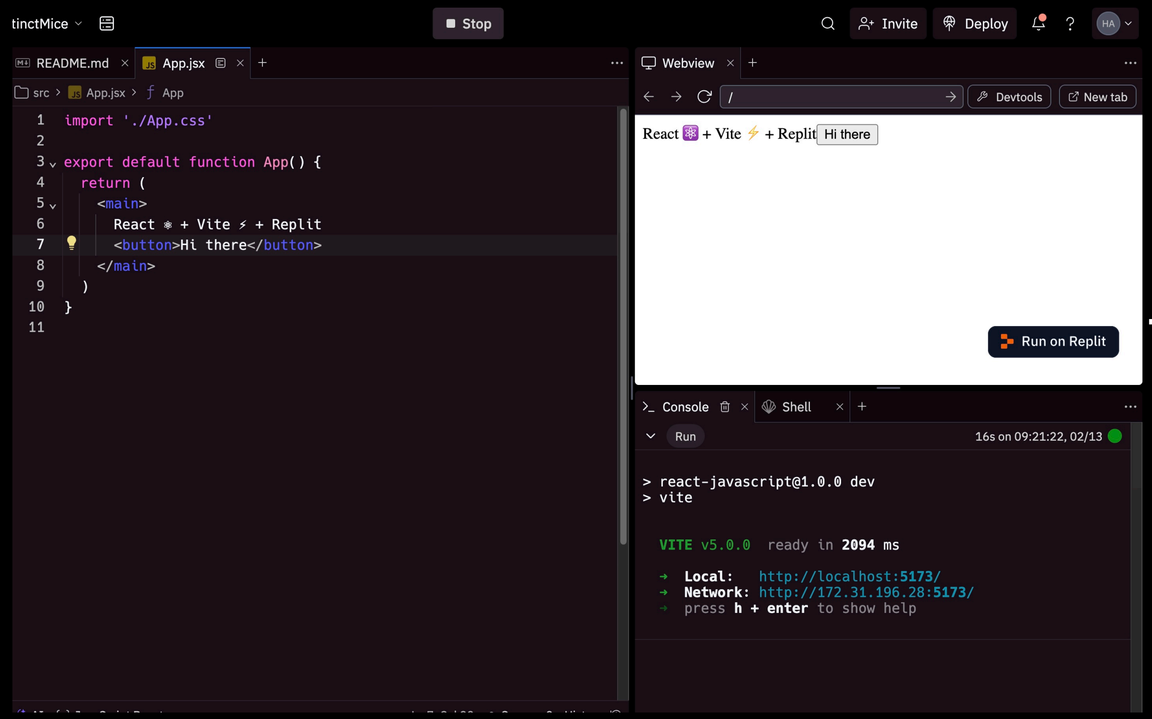
Execution service





Streaming logs

Execution service



Disconnects

Execute

Clean up resources

1. **Wait for a bit before removing the folder**
2. **Flush to S3**
3. **Stop any lingering process**

You can wait for a bit before doing this

Downsides

1. Remote code execution
2. Single server setup, doesn’t autoscale
3. Port conflicts between two users (every user is sharing resources on the same server)
4. Terminal is very ad-hoc/first principles

Code

Disclaimers

1. I’m using Node.js. Keep it simple
2. I’m using socket.io. Keep it simple
3. I’ll be writing code in TS, but nothing too strict. Keep it simple
4. I will not be adding any extra fluff that’s not needed for this tutorial (eslint, prettier).
5. No monorepos - code repetition

Should you create a zig based well linted 100% tested CI/CD implemented system?
Maybe

But we’re limiting the scope of this tutorial

Introducing PTY

If you want to do a terminal inside a browser
(or create your own terminal for eg)
What you can do is create a `pseudo terminal`
that your browser can talk to

xterm.js is a library that lets you fetch and
forward keystrokes to a `pseudo terminal` that
you can spawn on your server

https://github.com/xtermjs/xterm.js

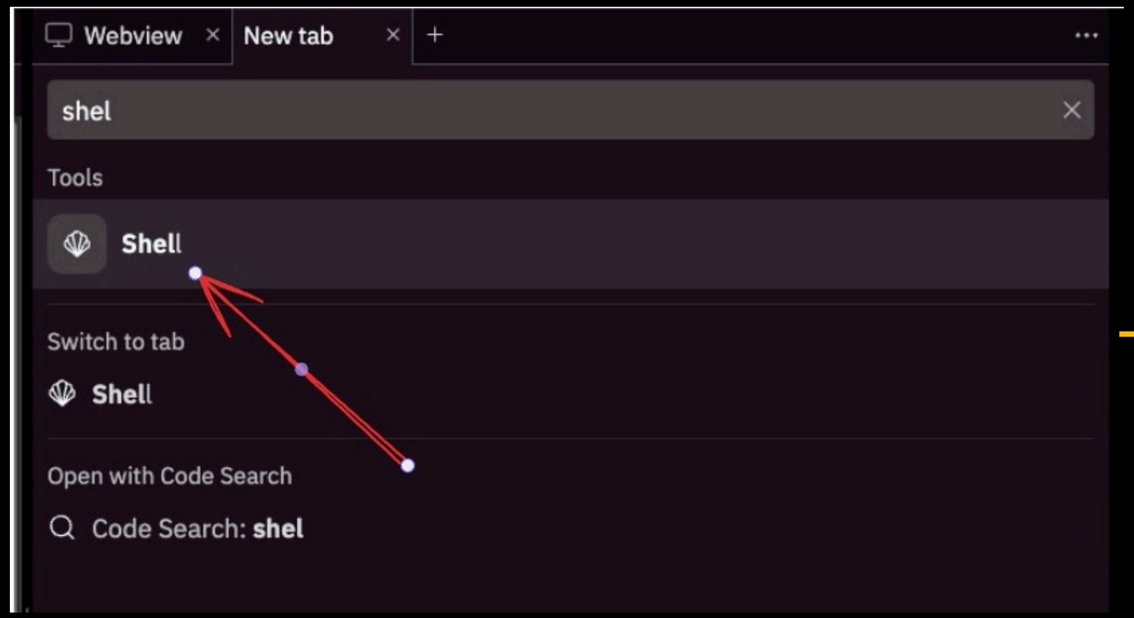
https://github.com/replit/ruspty

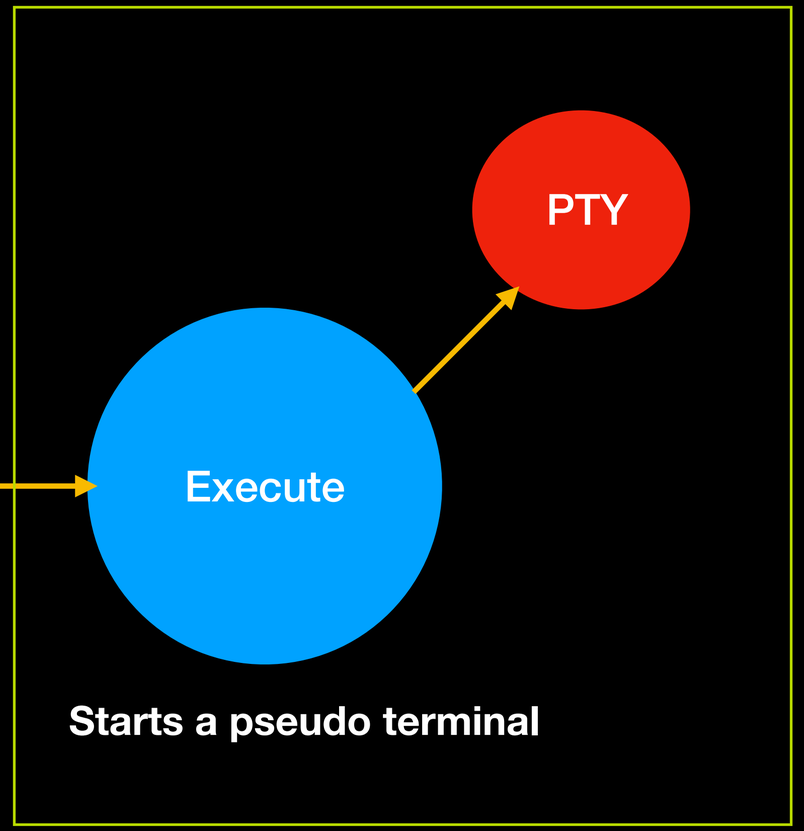
Introducing PTY

Old approach

Introducing PTY

New approach

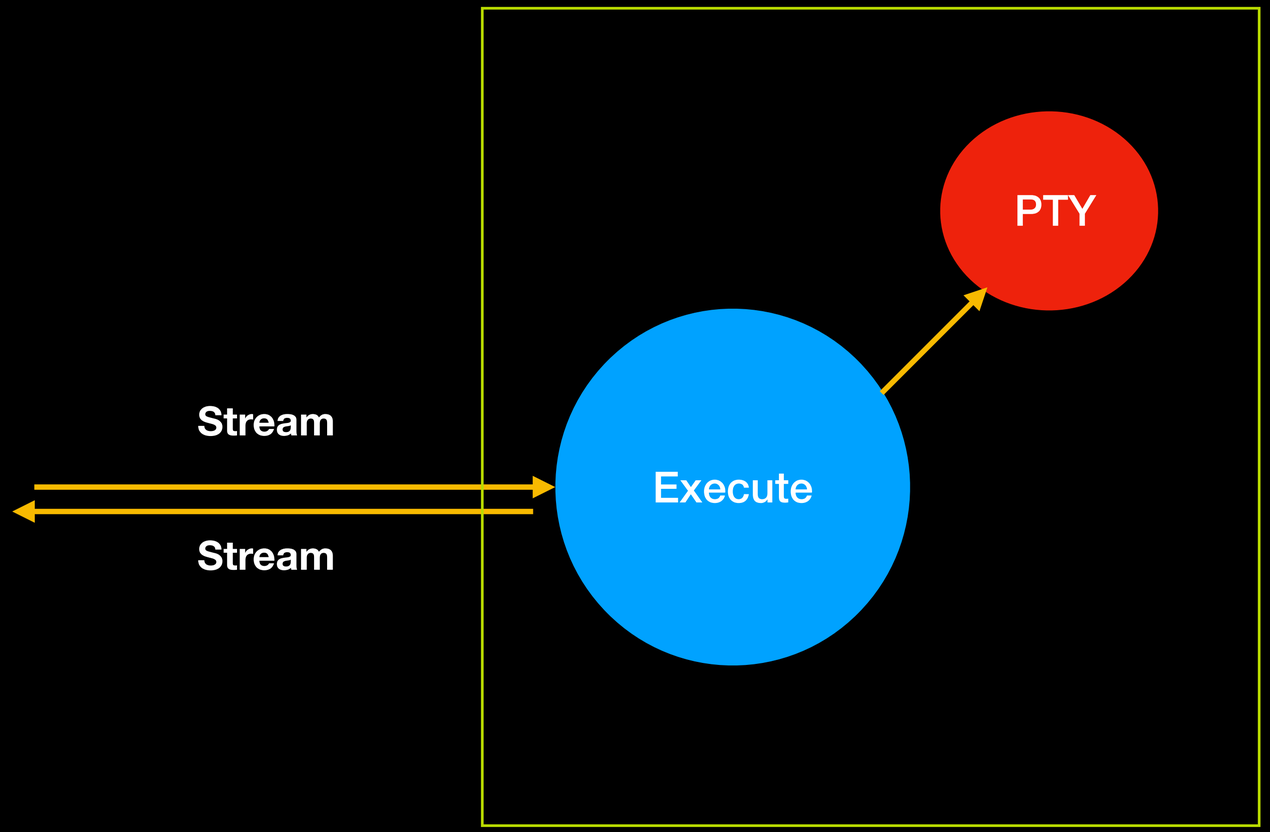




Introducing PTY

New approach

```
# llss
app         cron                  Gemfile       log         spec
bin         db                    Gemfile.lock  public      tmp
config      docker-entrypoint.sh  lib           Rakefile    vendor
config.ru   docs                  LICENSE       scripts
#
app         cron                  Gemfile       log         spec
bin         db                    Gemfile.lock  public      tmp
config      docker-entrypoint.sh  lib           Rakefile    vendor
config.ru   docs                  LICENSE       scripts
# ■
```



Part 2 | The good solution

Part 2 | The good solution

What were the biggest problems in approach #1 ?
1. Remote code execution isn’t safe
2. Doesn’t autoscale

Part 2 | The good solution

There are two approaches you can take

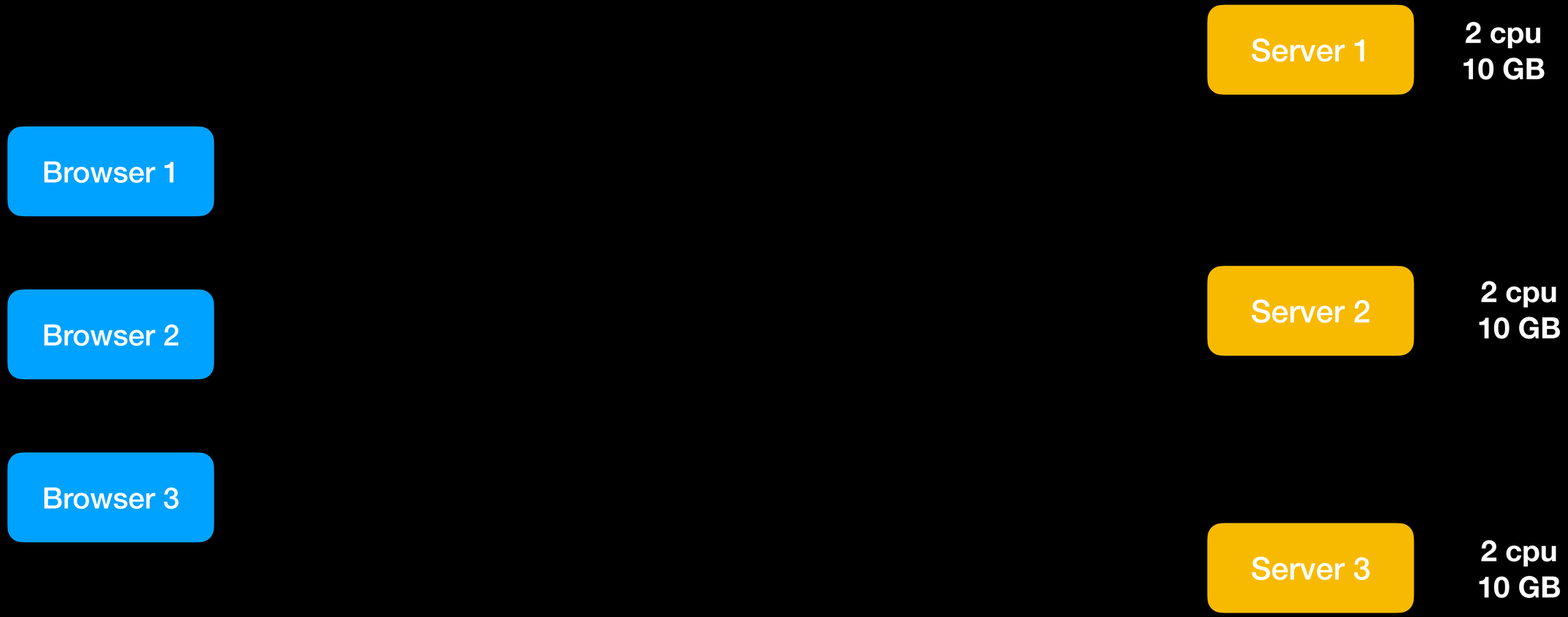
Cloud specific Autoscaling

AWS - Auto scaling Groups, ECS

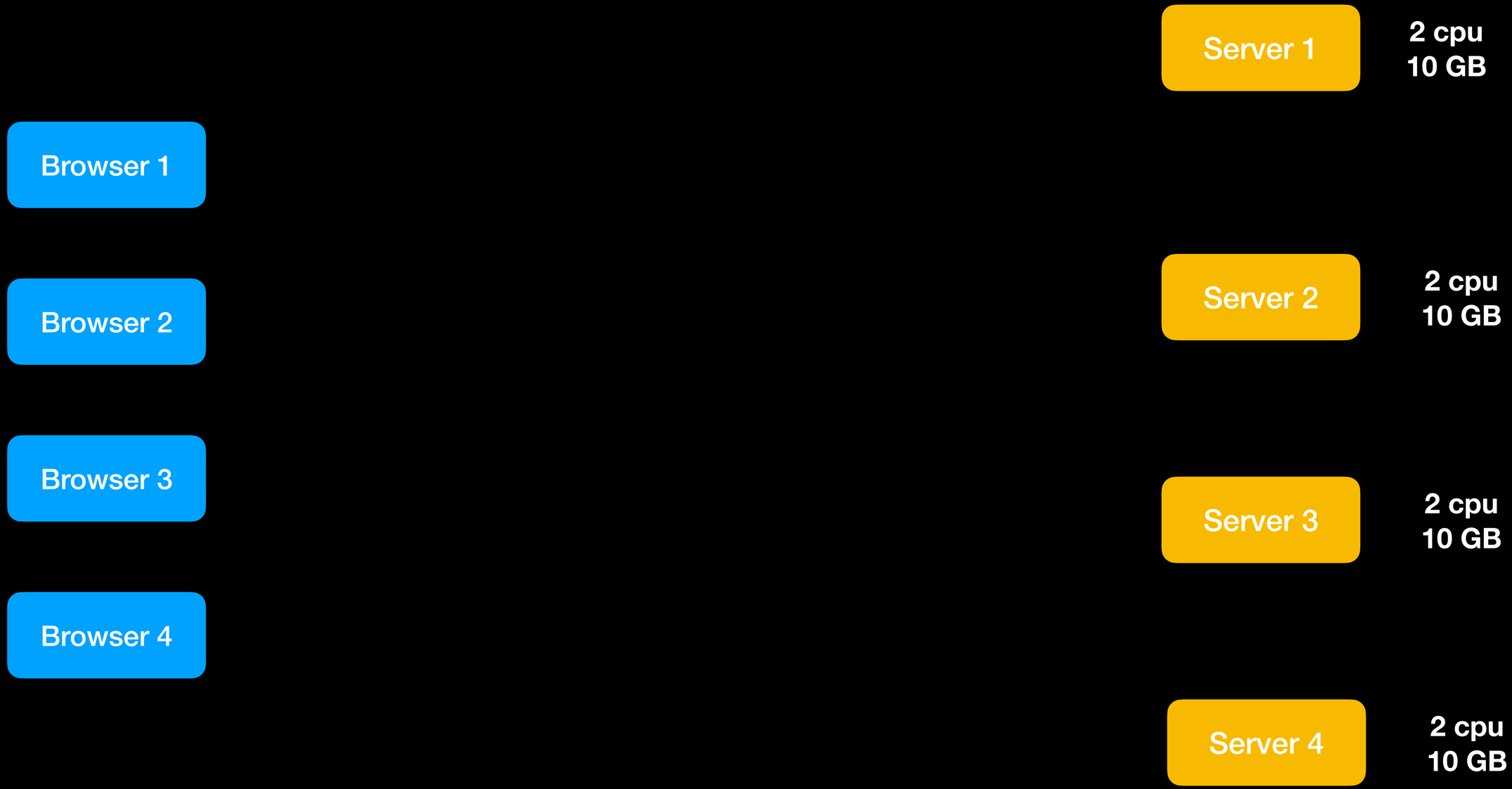
Container Orchestration

Kubernetes

Cloud specific Autoscaling



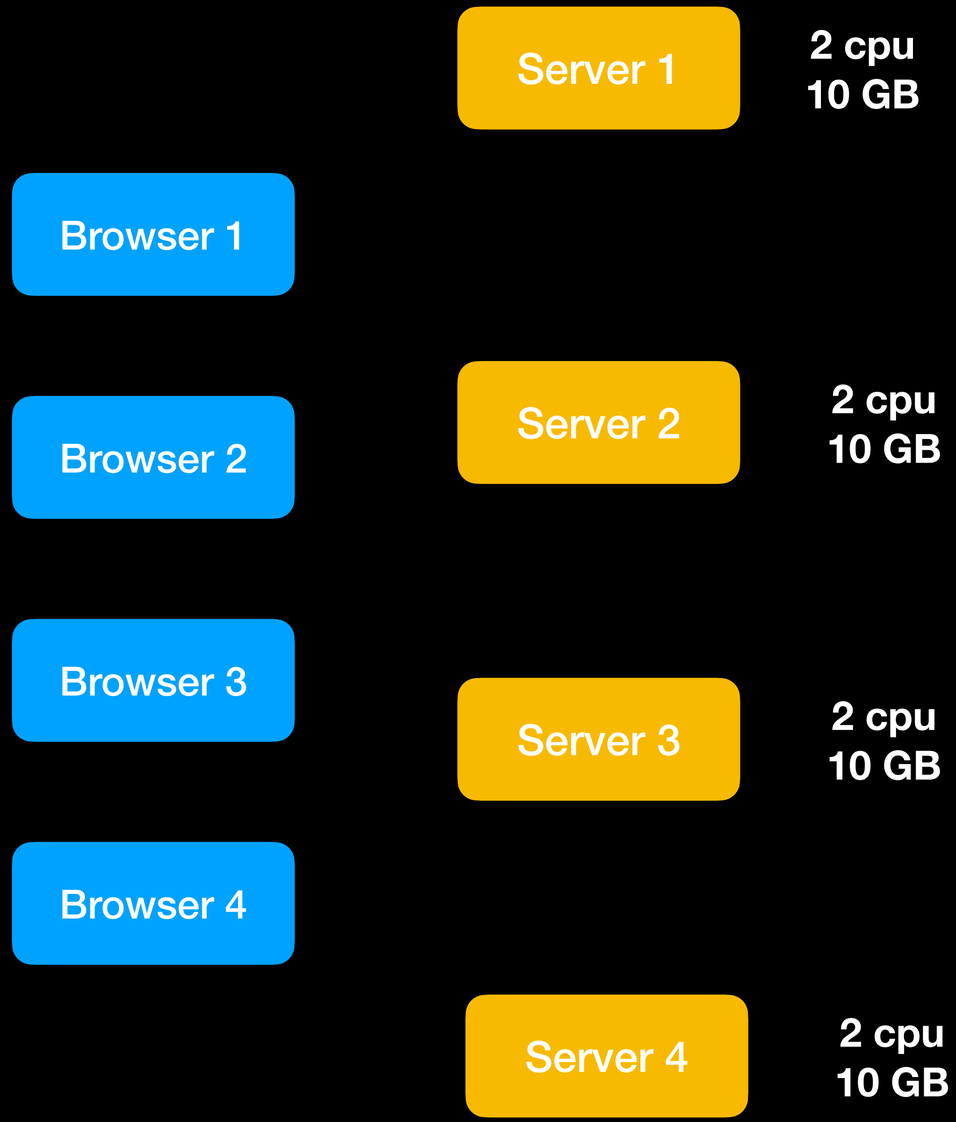
Cloud specific Autoscaling



Cloud specific Autoscaling

Upsides
1. Easy to do
2. Provides you a way to securely run code
3. Autoscales
4. No port conflicts

Downsides
1. Bootup time (not a huge problem)
2. Over provisioned servers
3. Not cloud agnostic



Kubernetes

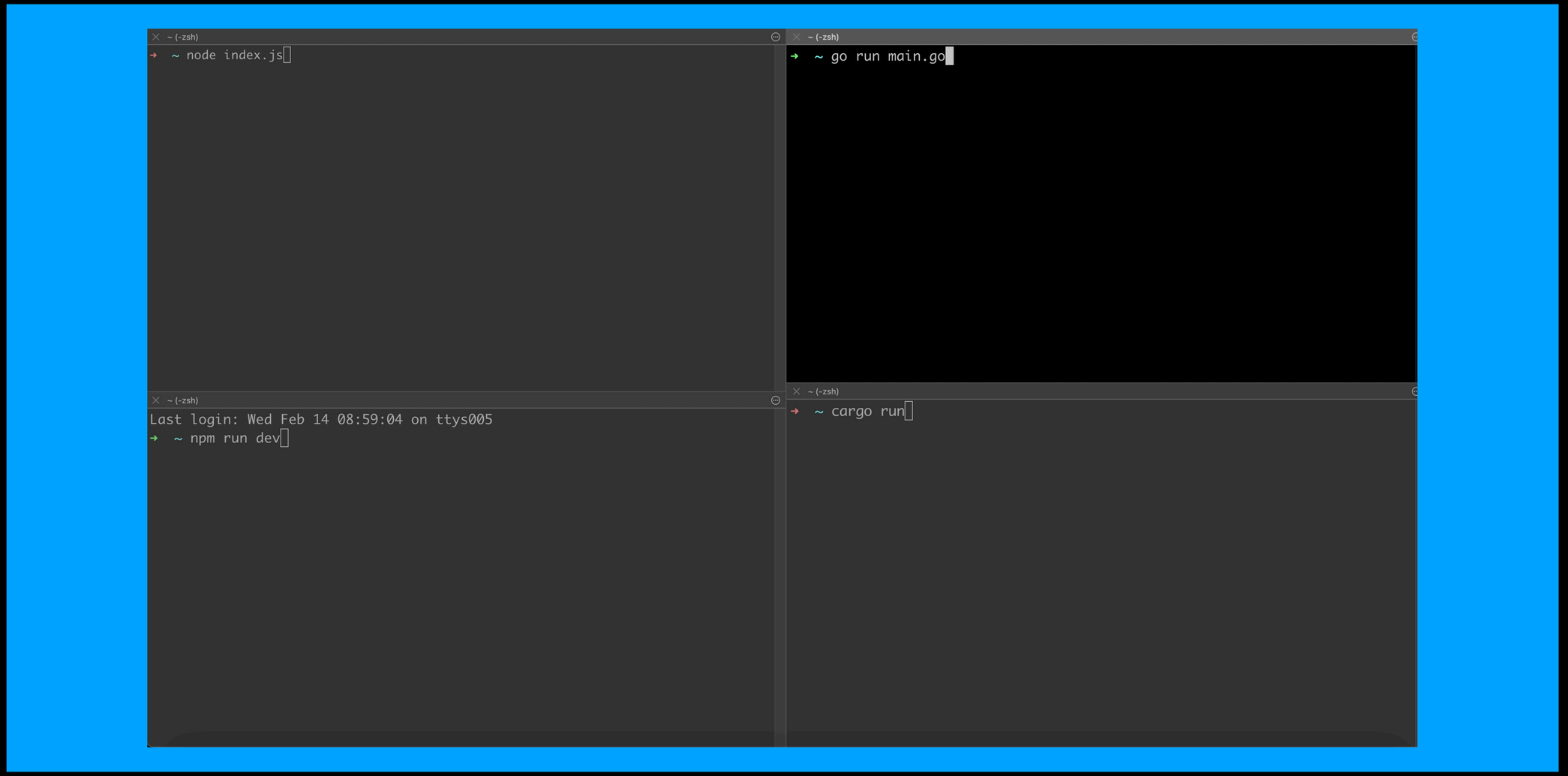
Before we get into why this approach is good, let’s understand what

1. Containers are
2. k8s is

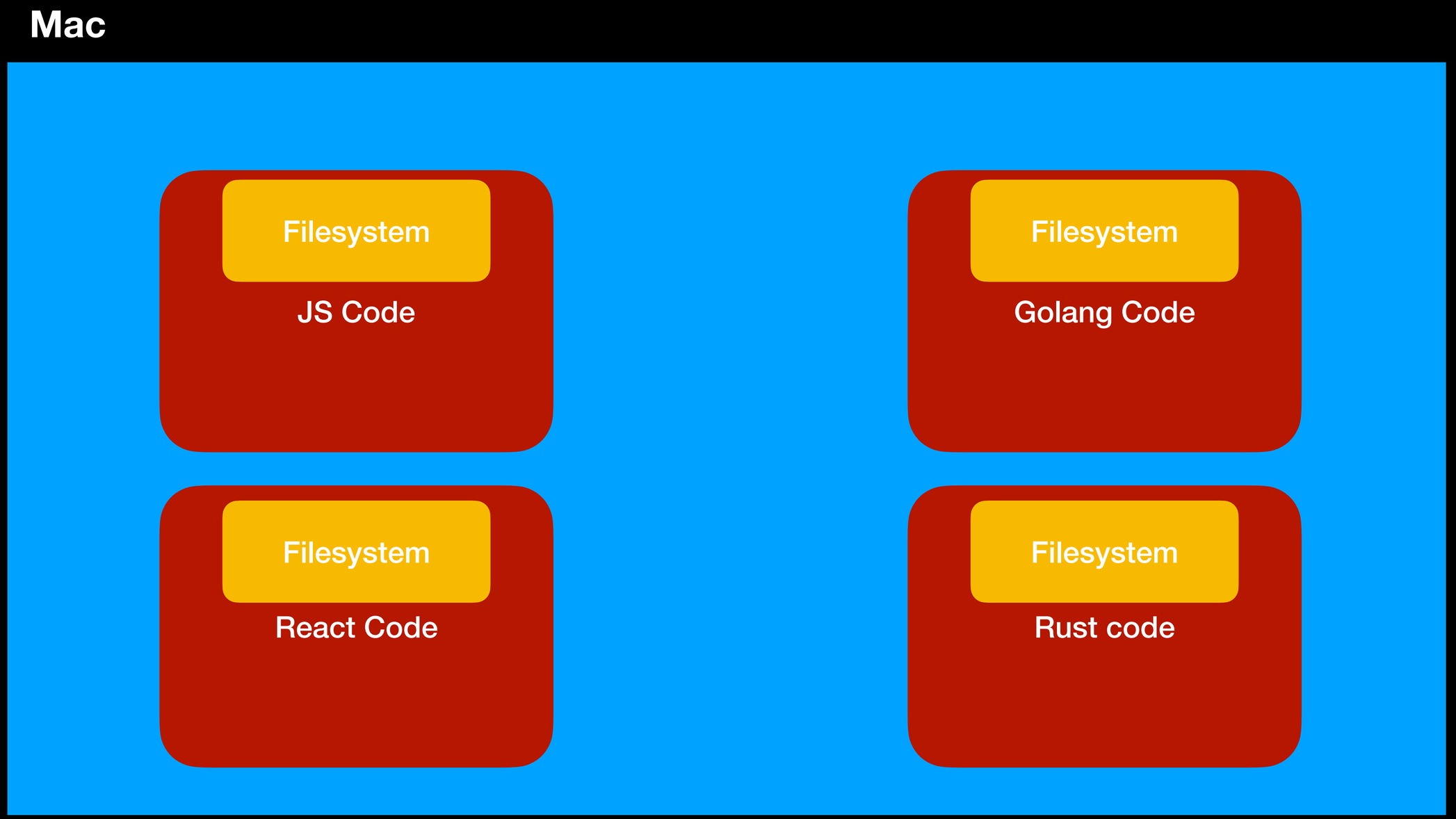
Containers

Containers

Mac

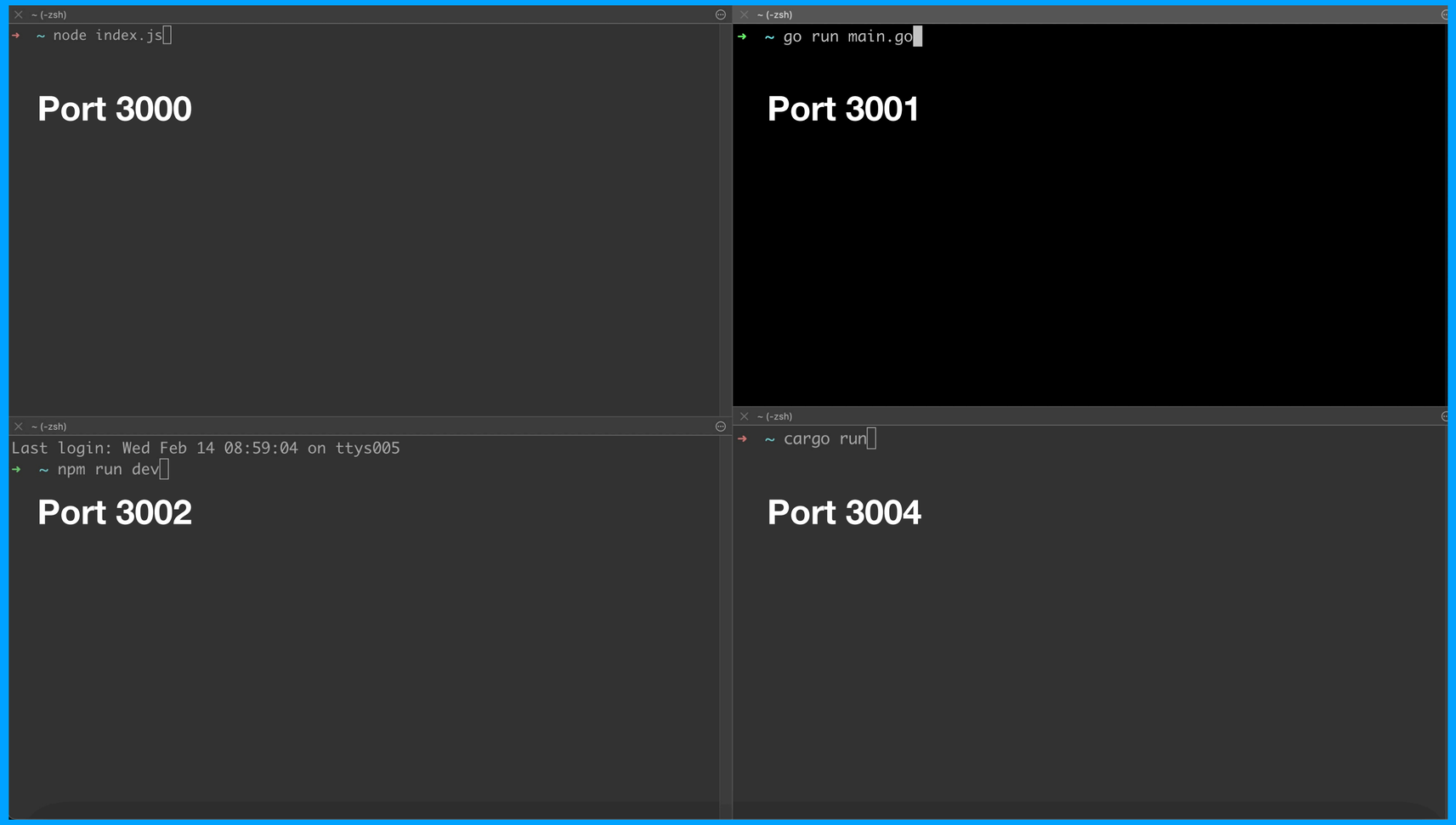


Containers

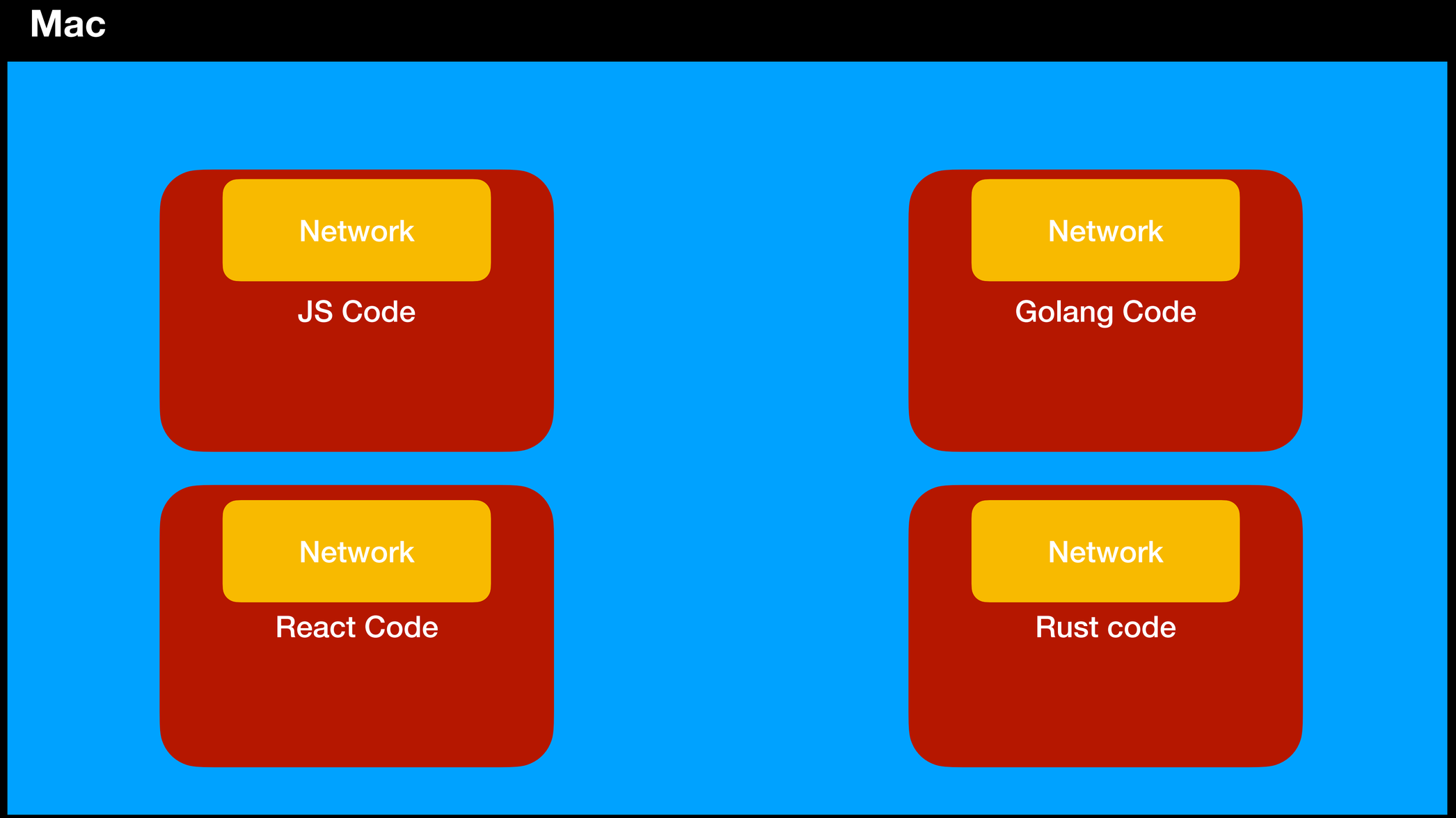

Mac
Filesystem
JS Code
Filesystem
Golang Code
Filesystem
React Code
Filesystem
Rust code

Containers

Mac



Containers


Mac
Network
JS Code
Network
Golang Code
Network
React Code
Network
Rust code

Container Orchestration - Kubernetes

What if you want to run multiple such containers,
and autoscale your servers?

Container Orchestration - Kubernetes

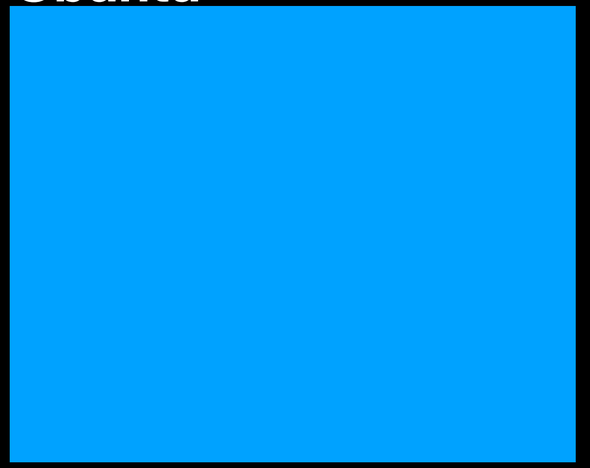
What if you could have a cluster of machines and describe in a single file how to run multiple containers

Mac

Linux

Windows

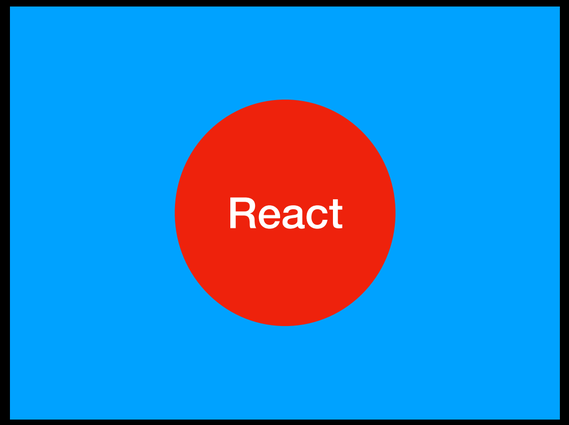
Ubuntu



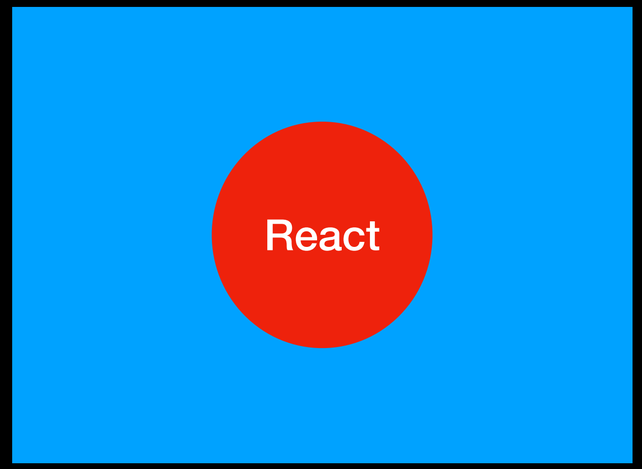
Container Orchestration - Kubernetes

What if you could have a cluster of machines and describe in a single file how to run multiple containers

Mac

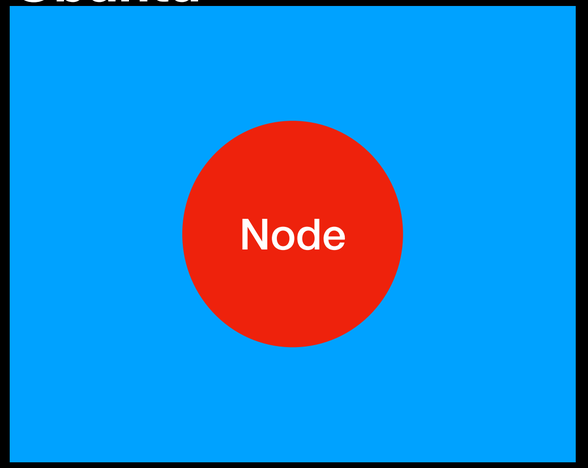


Linux



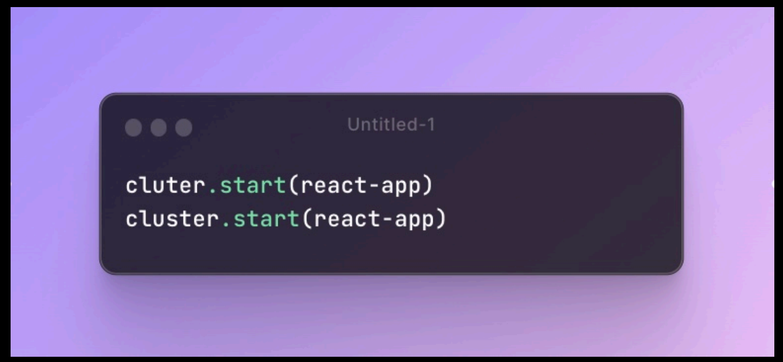
Windows

Ubuntu

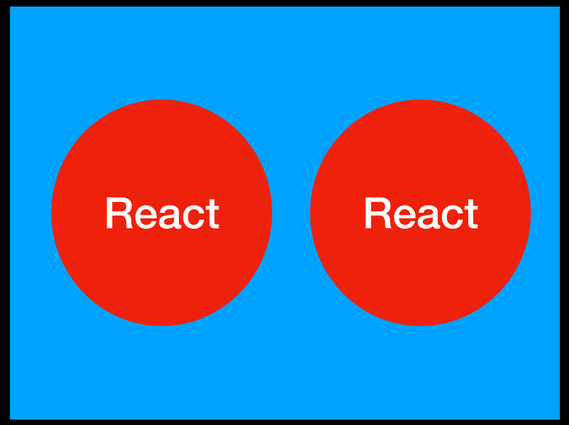


Container Orchestration - Kubernetes

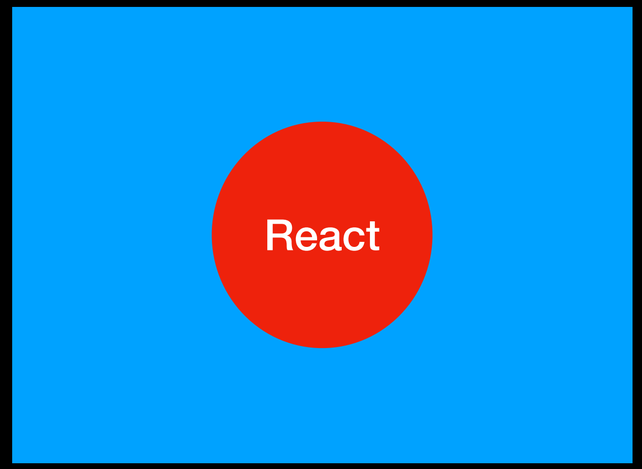
Not just a file, you could add and remove containers by doing a function call



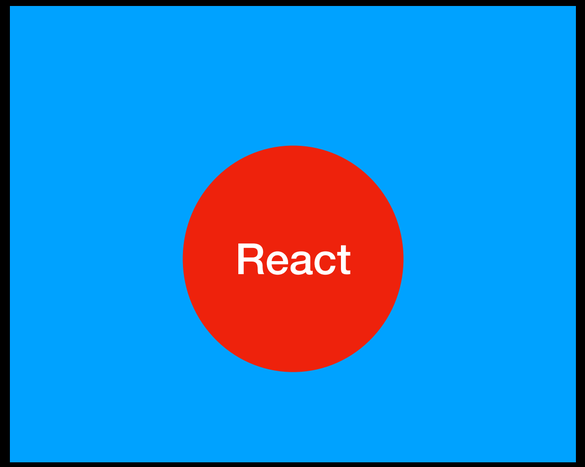
Mac



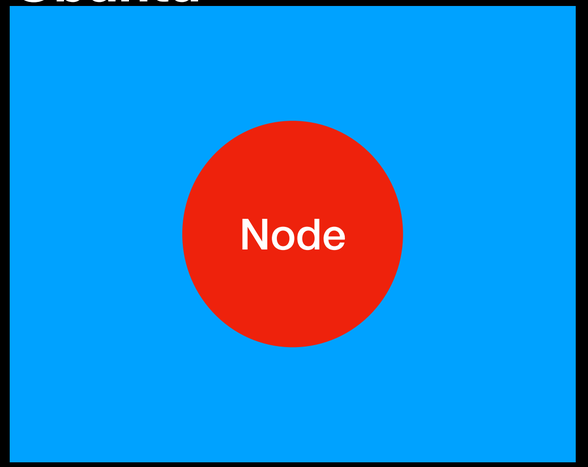
Linux



Windows



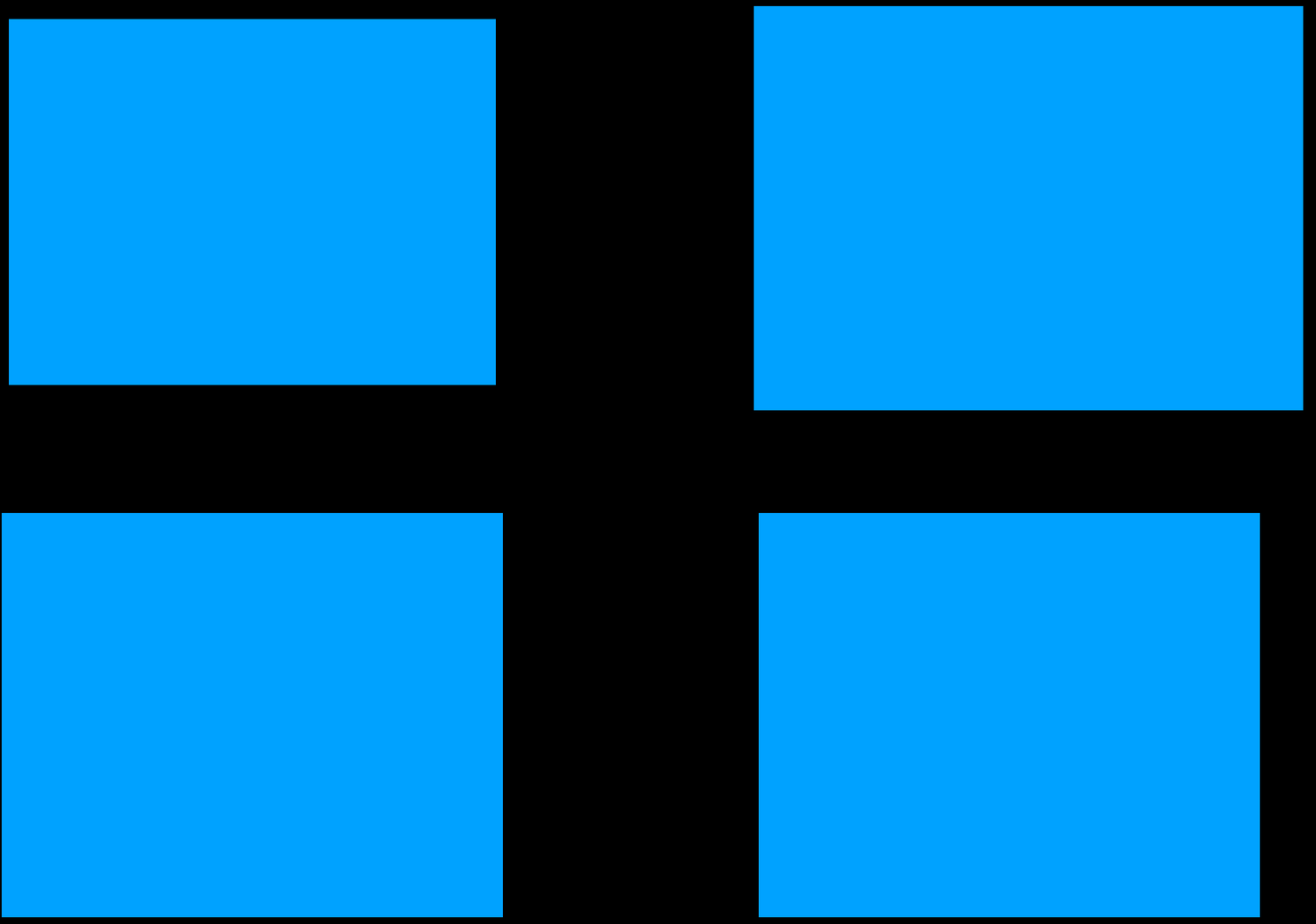
Ubuntu



Container Orchestration - Kubernetes

Some Kubernetes jargon

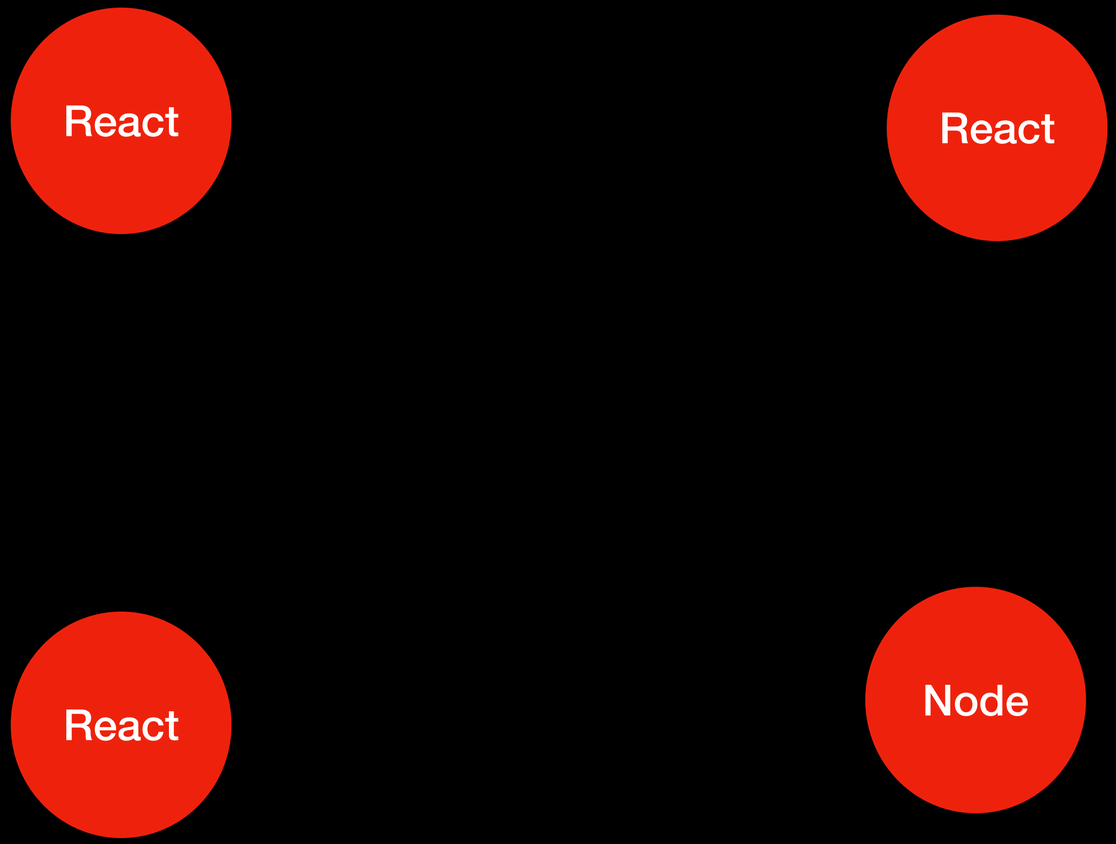
1. Nodes



Container Orchestration - Kubernetes

Some Kubernetes jargon

1. Nodes
2. Pods



Container Orchestration - Kubernetes

Some Kubernetes jargon

1. Nodes
2. Pods
3. Services
4. Ingress

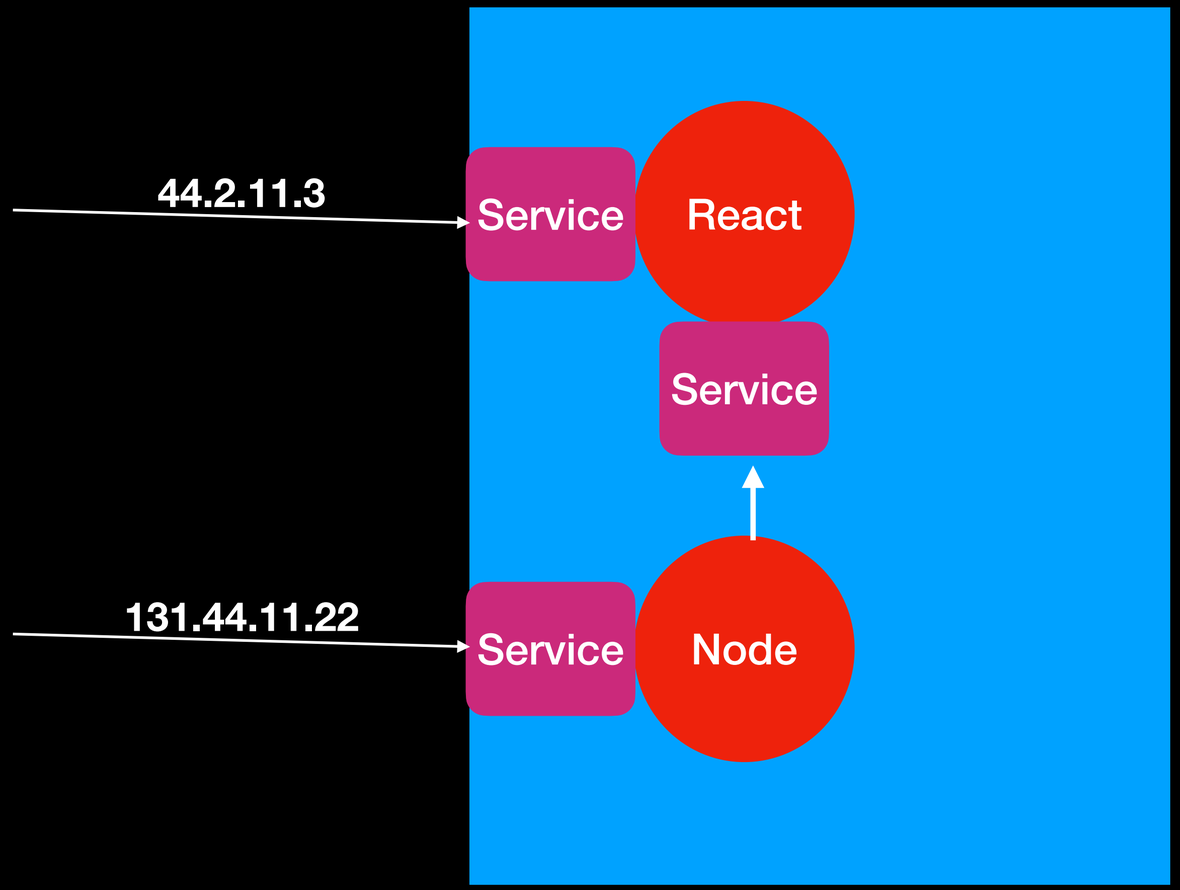
Container Orchestration - Kubernetes

Services and ingress control how your application is exposed
How can people access your services running inside pods?
How does a user that has created a repl, reach the container where their code is present?

Container Orchestration - Kubernetes

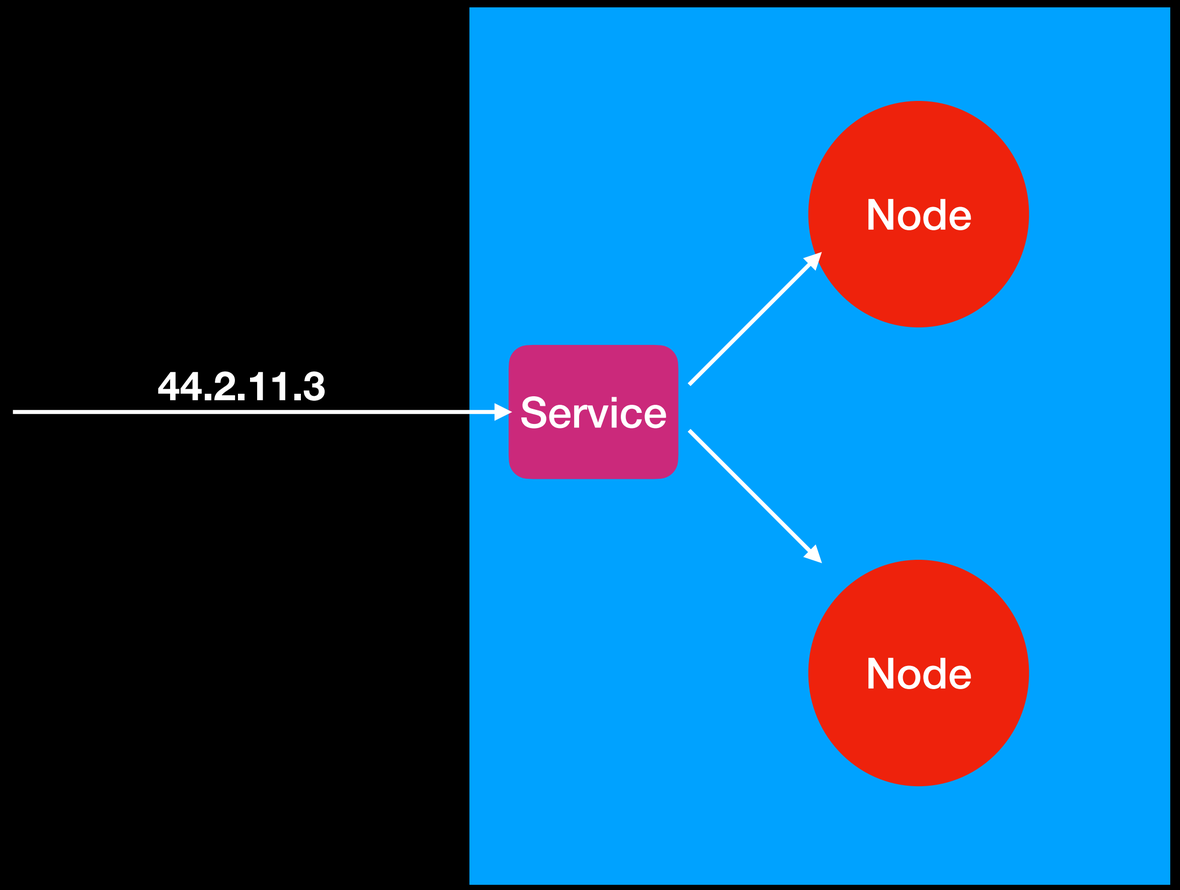
Services let you expose your pods

1. You can either expose them on the internet
2. Or you can let other people access your pods
3. Each pod can have an associated service



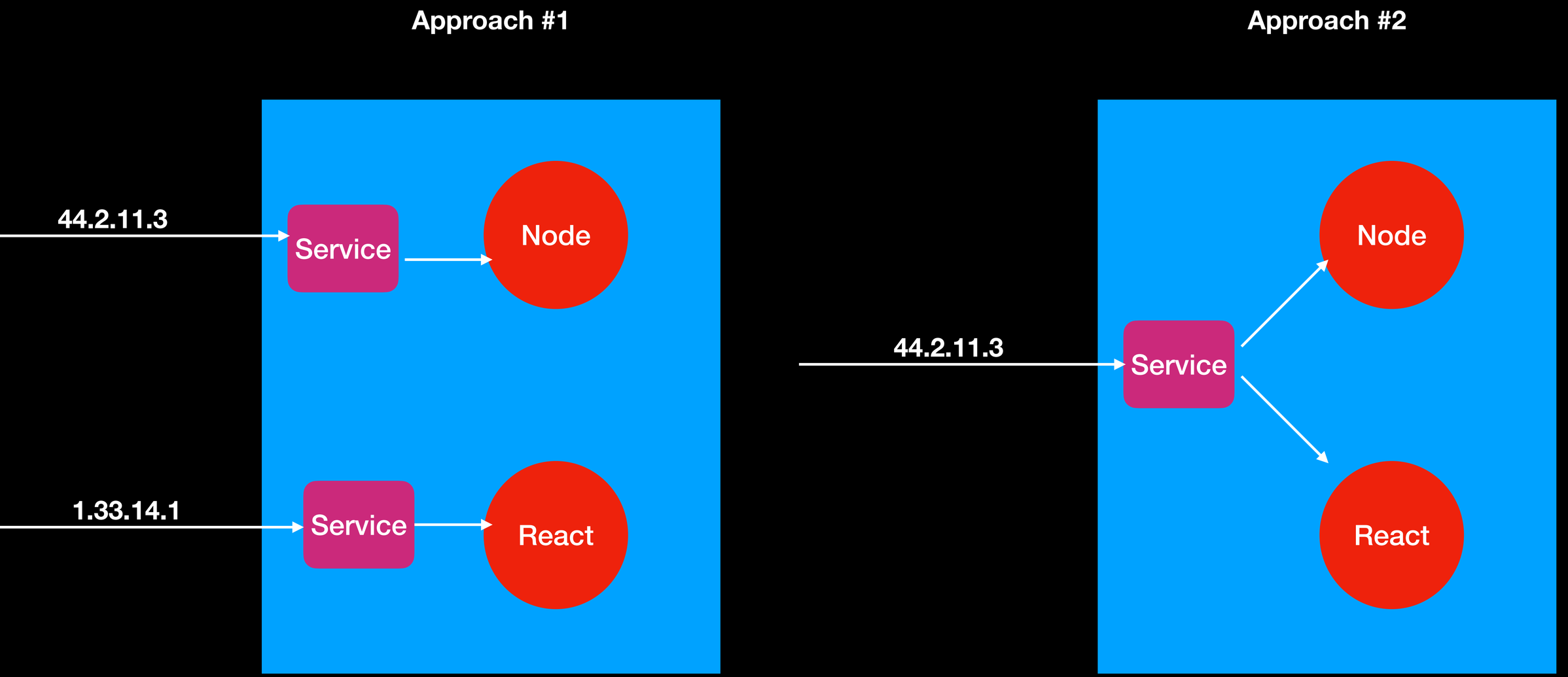
Container Orchestration - Kubernetes

A service can also load balance across pods



Container Orchestration - Kubernetes

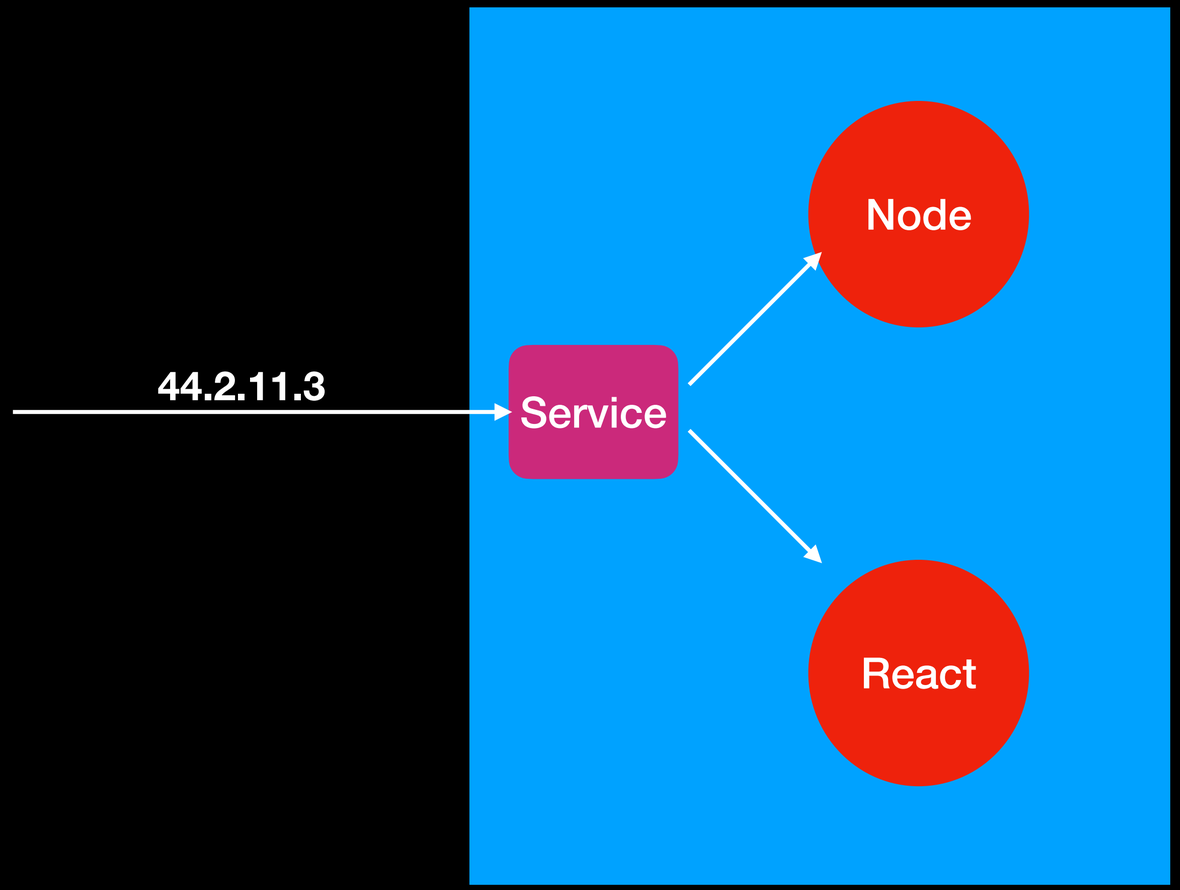
Let’s go through the replit runner and see
how they expose services
What do you think they do?



Container Orchestration - Kubernetes

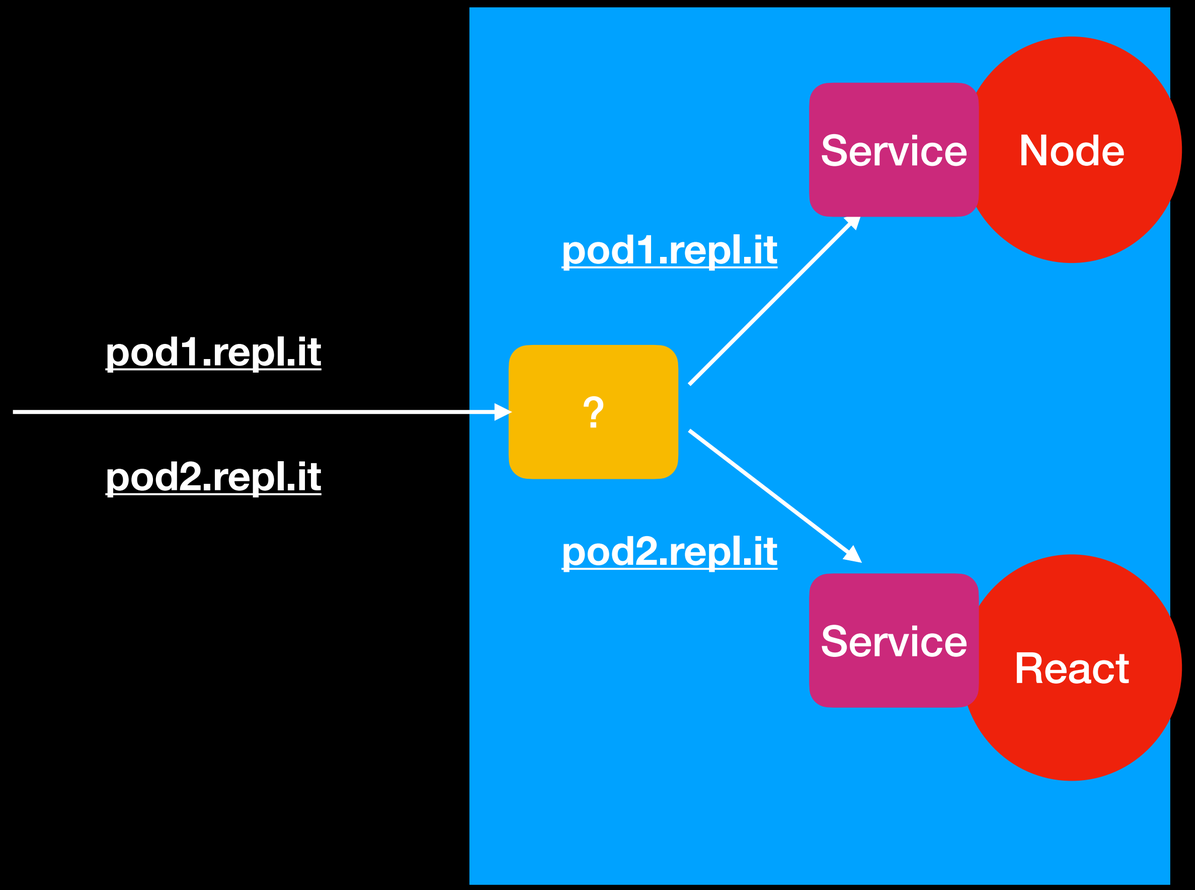
You might feel like they do this , and they do
But services don’t let you do path based routing

They will load balance, but they won’t let you control where the request should go based on the host URL



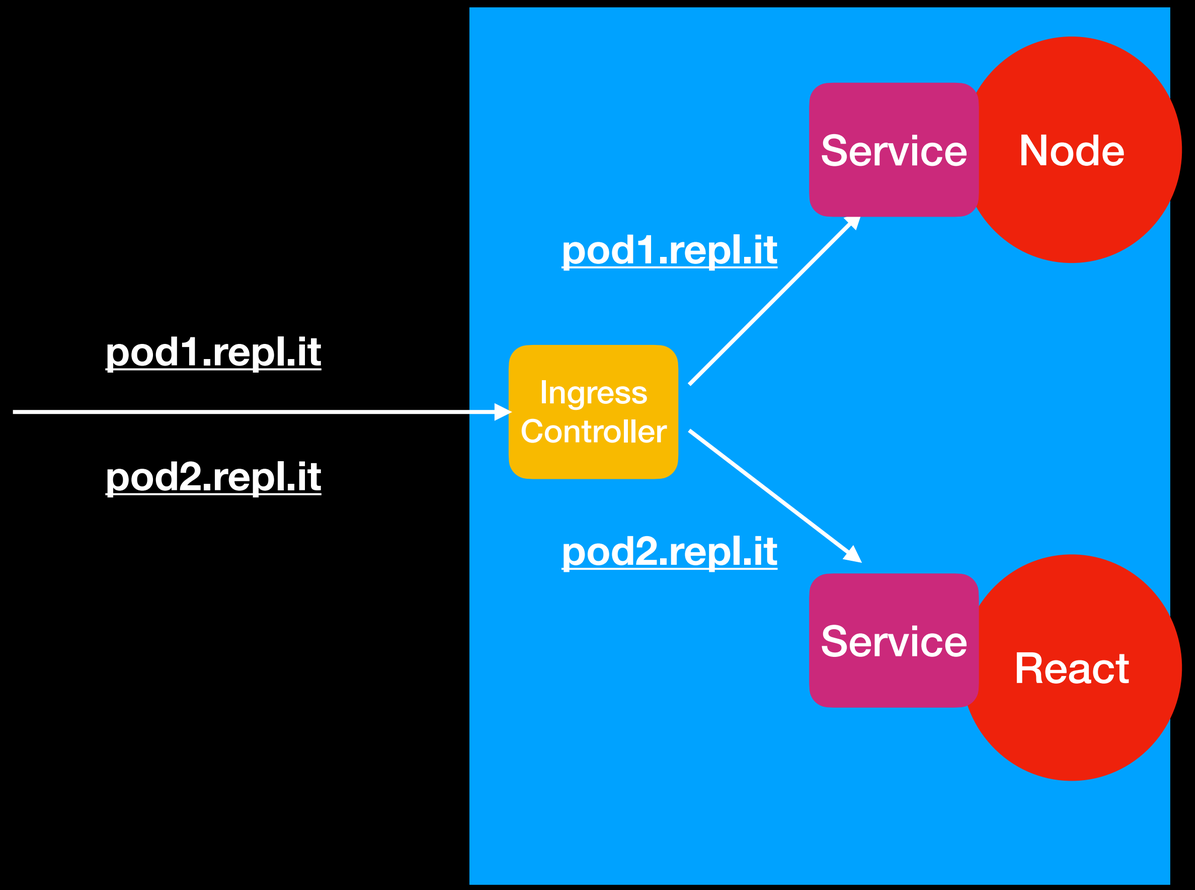
Container Orchestration - Kubernetes

How can we do path based routing?



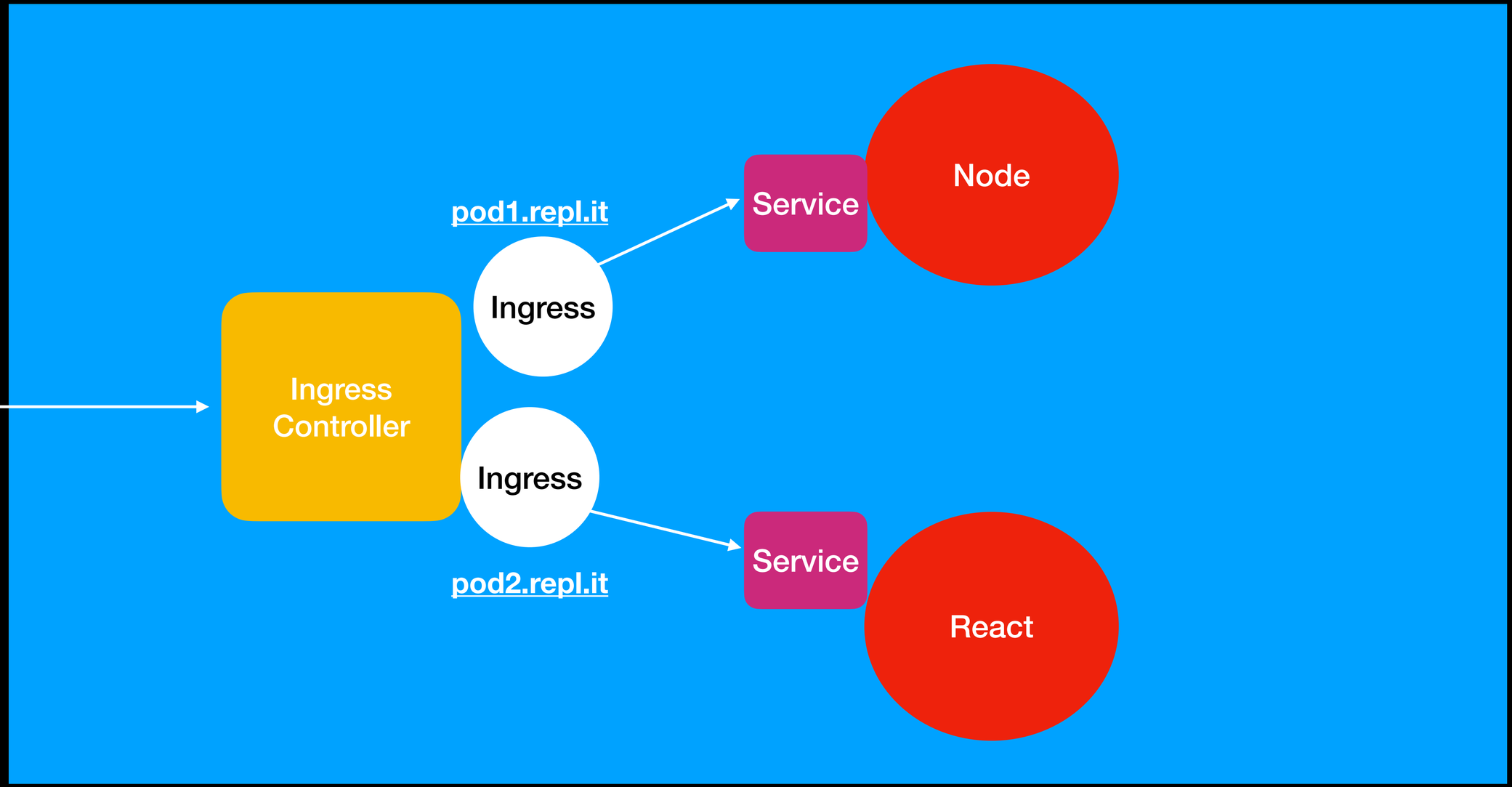
Container Orchestration - Kubernetes

Ingress



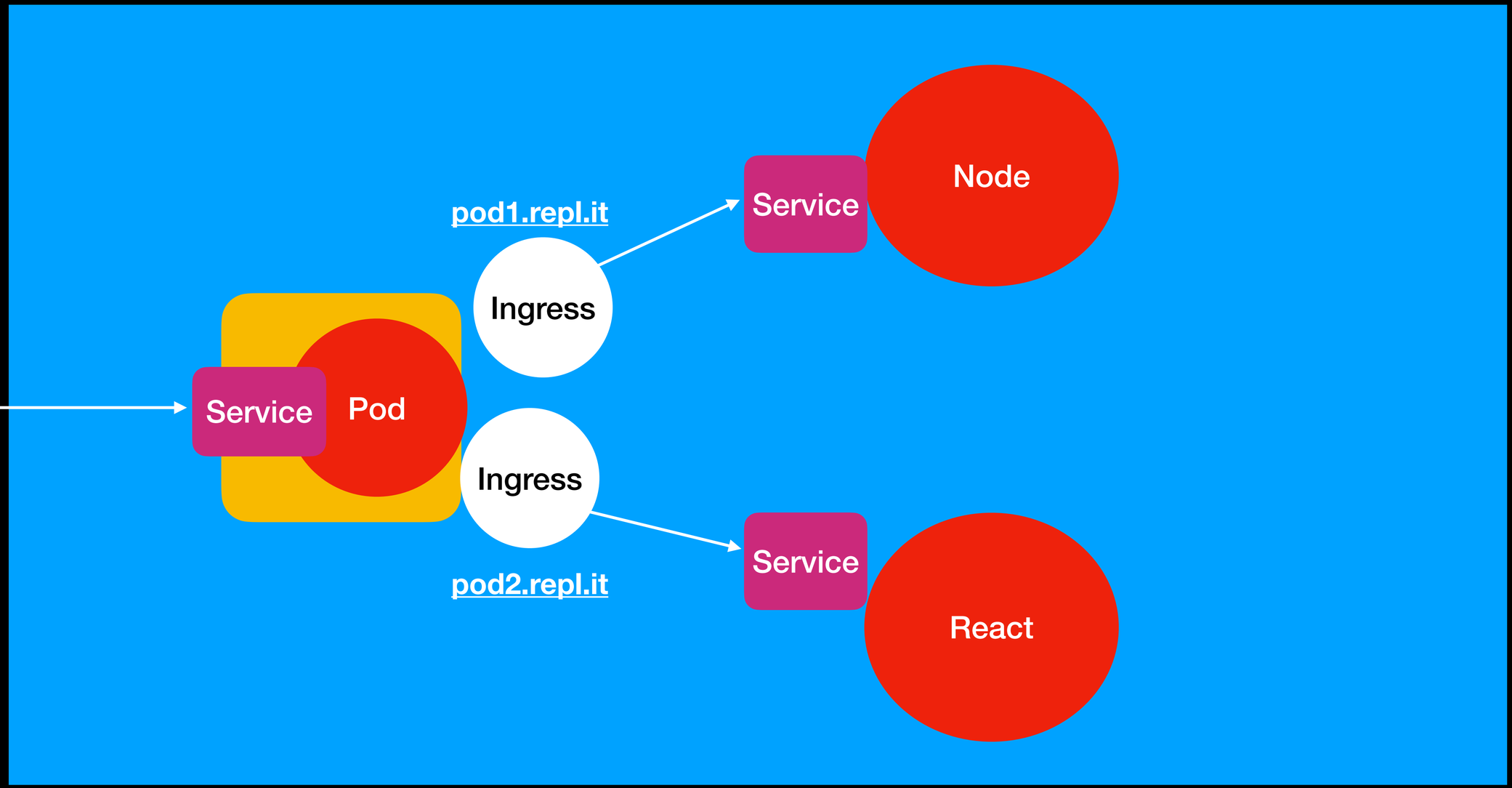
Container Orchestration - Kubernetes

Ingress Controller and Ingress



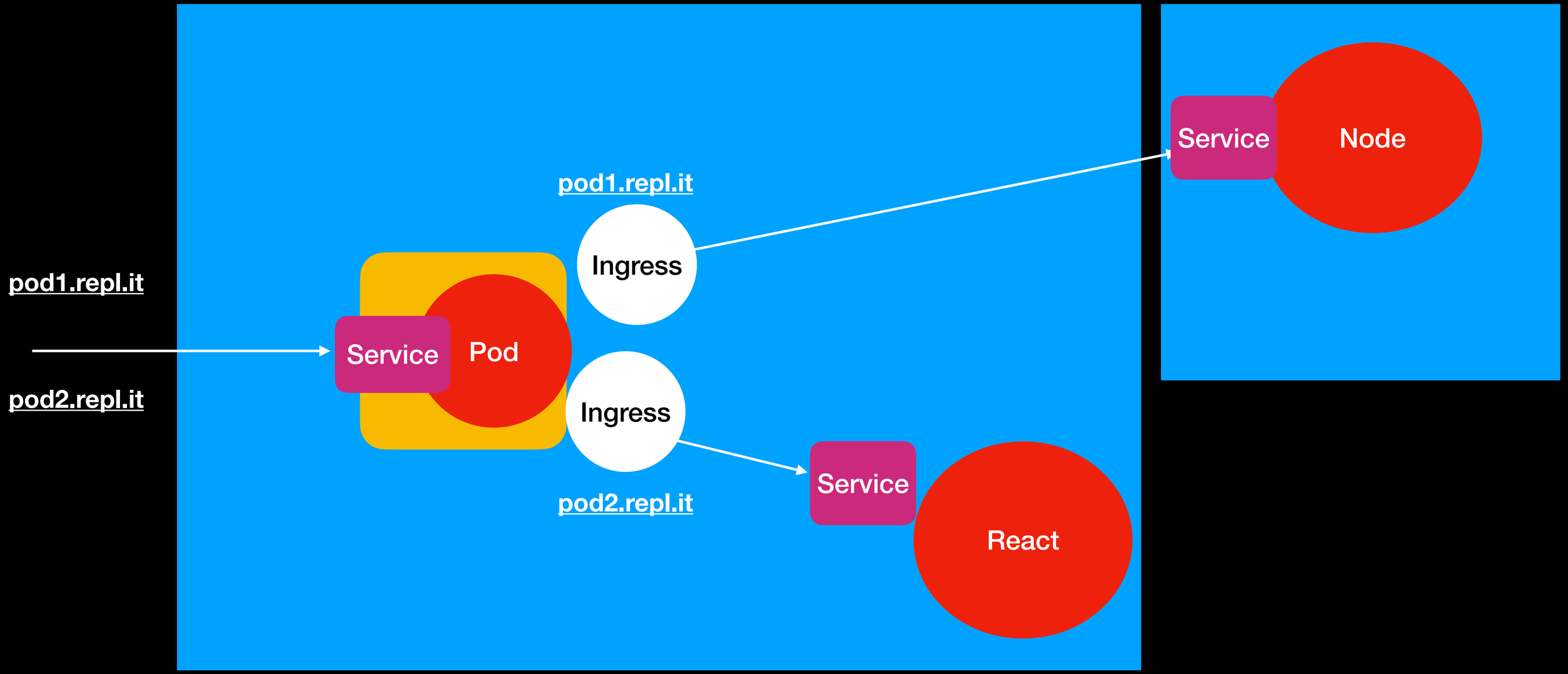
Container Orchestration - Kubernetes

Ingress Controller and Ingress



Container Orchestration - Kubernetes

They can exist on different nodes as well
As long as they are in the same cluster, Ingress controller should be able to route traffic

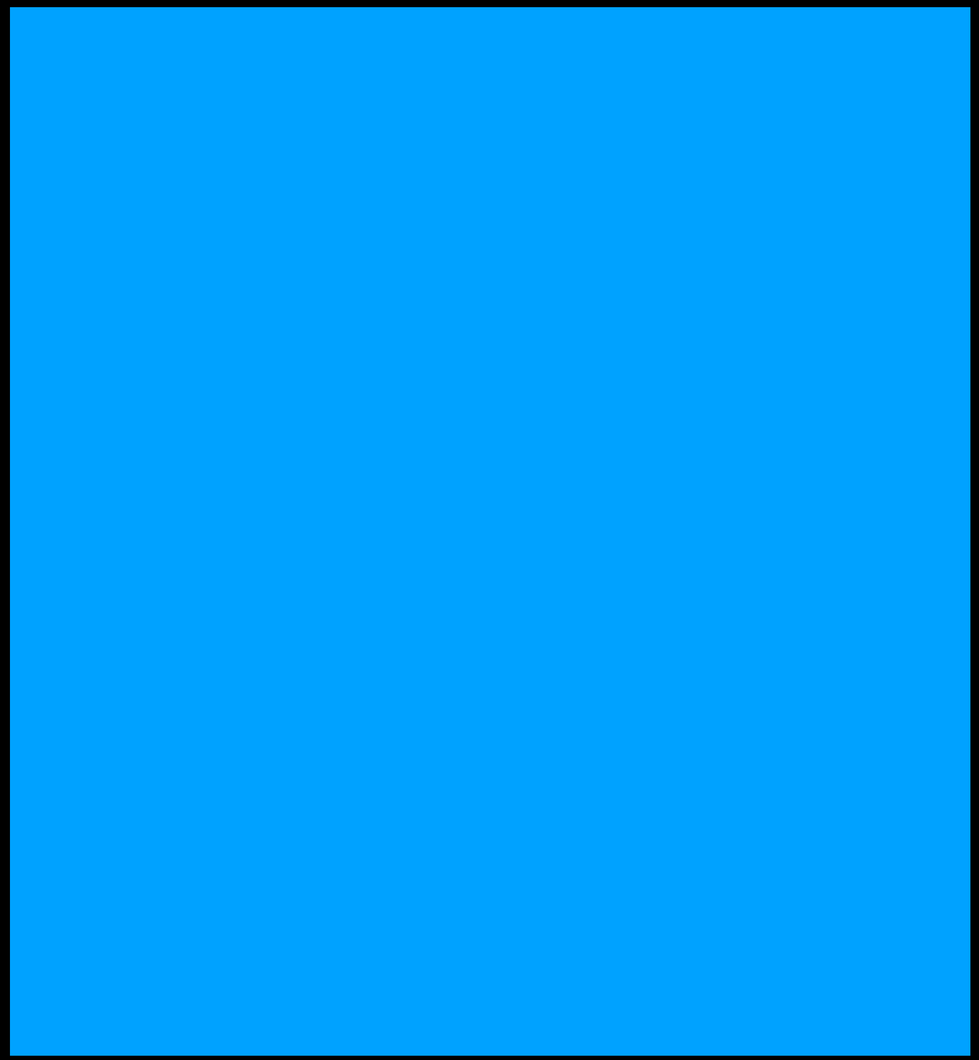


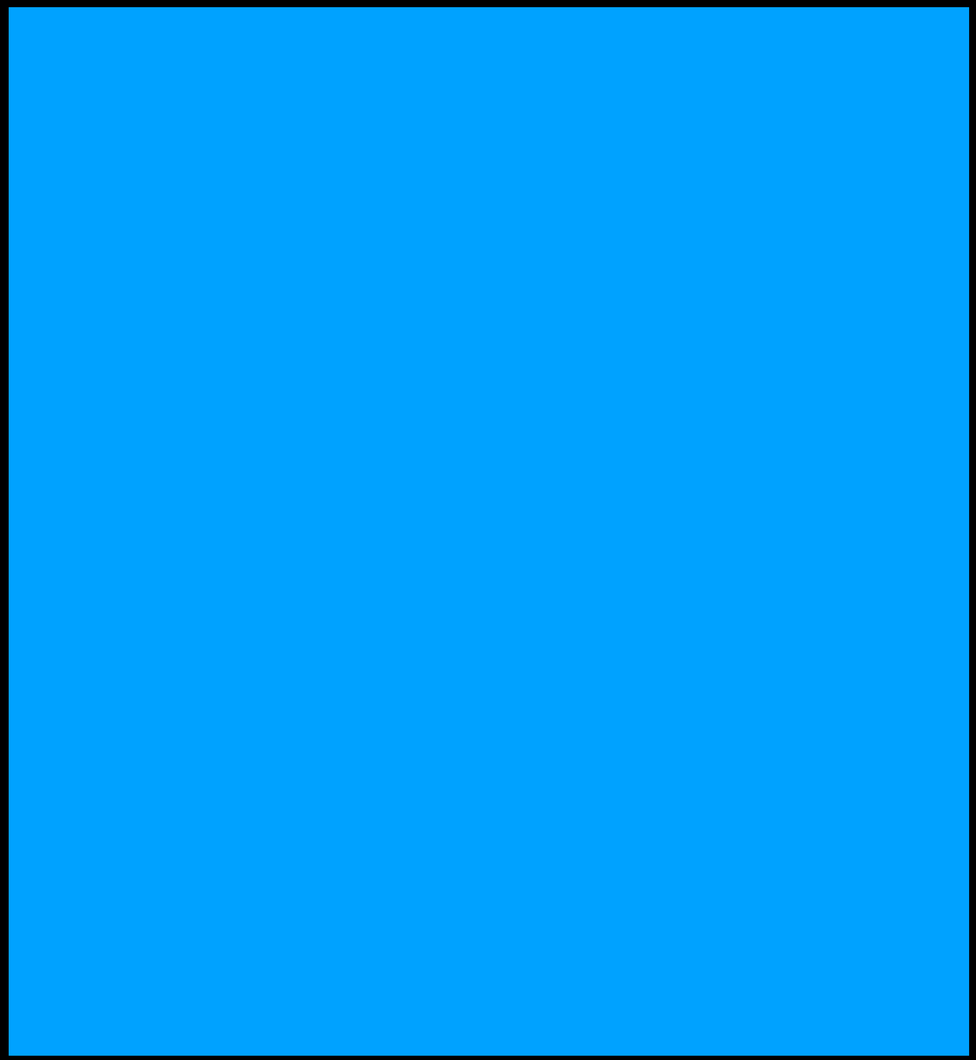
Container Orchestration - Kubernetes

Given all this information, can you guess the final architecture?

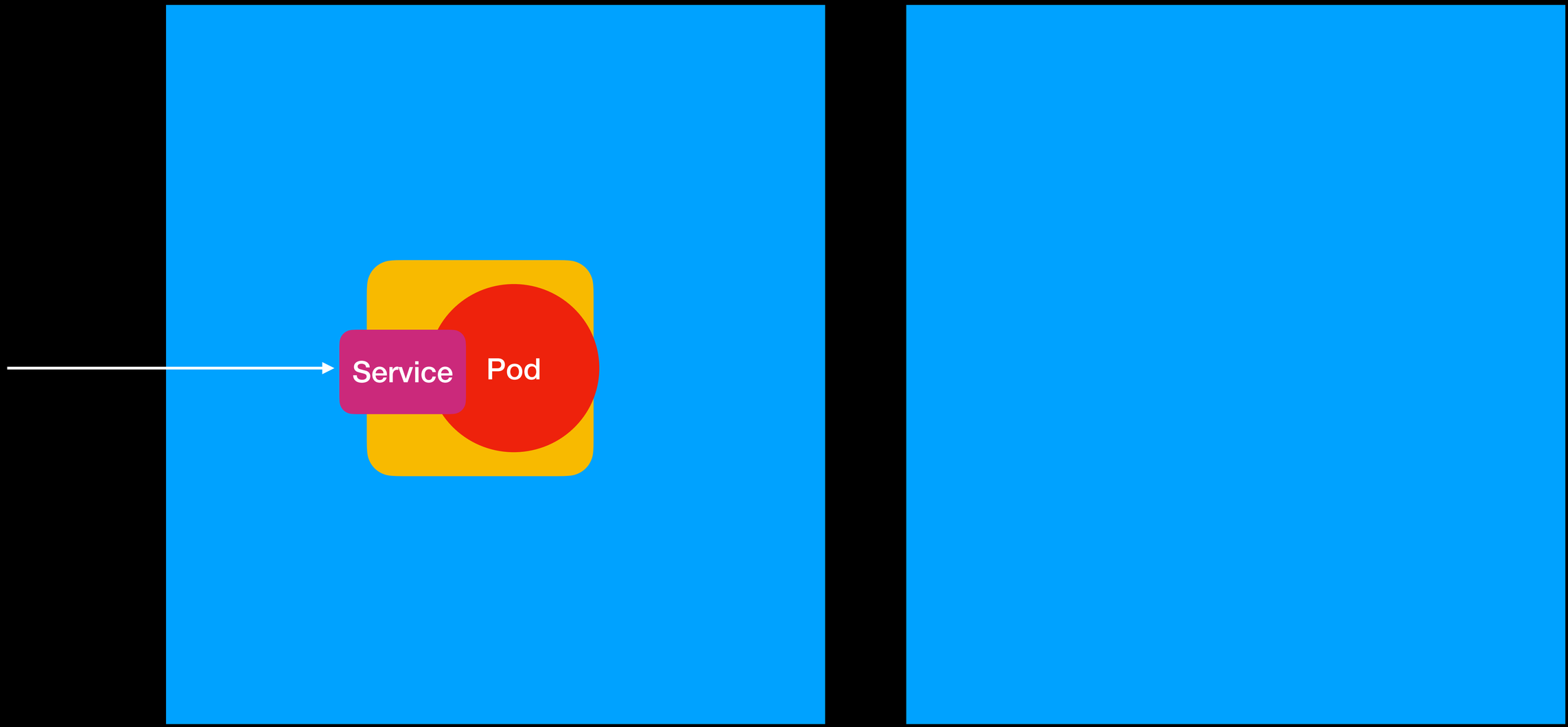
Container Orchestration - Kubernetes

Step 1 - Start a k8s cluster, set some autoscaling policies on the nodes

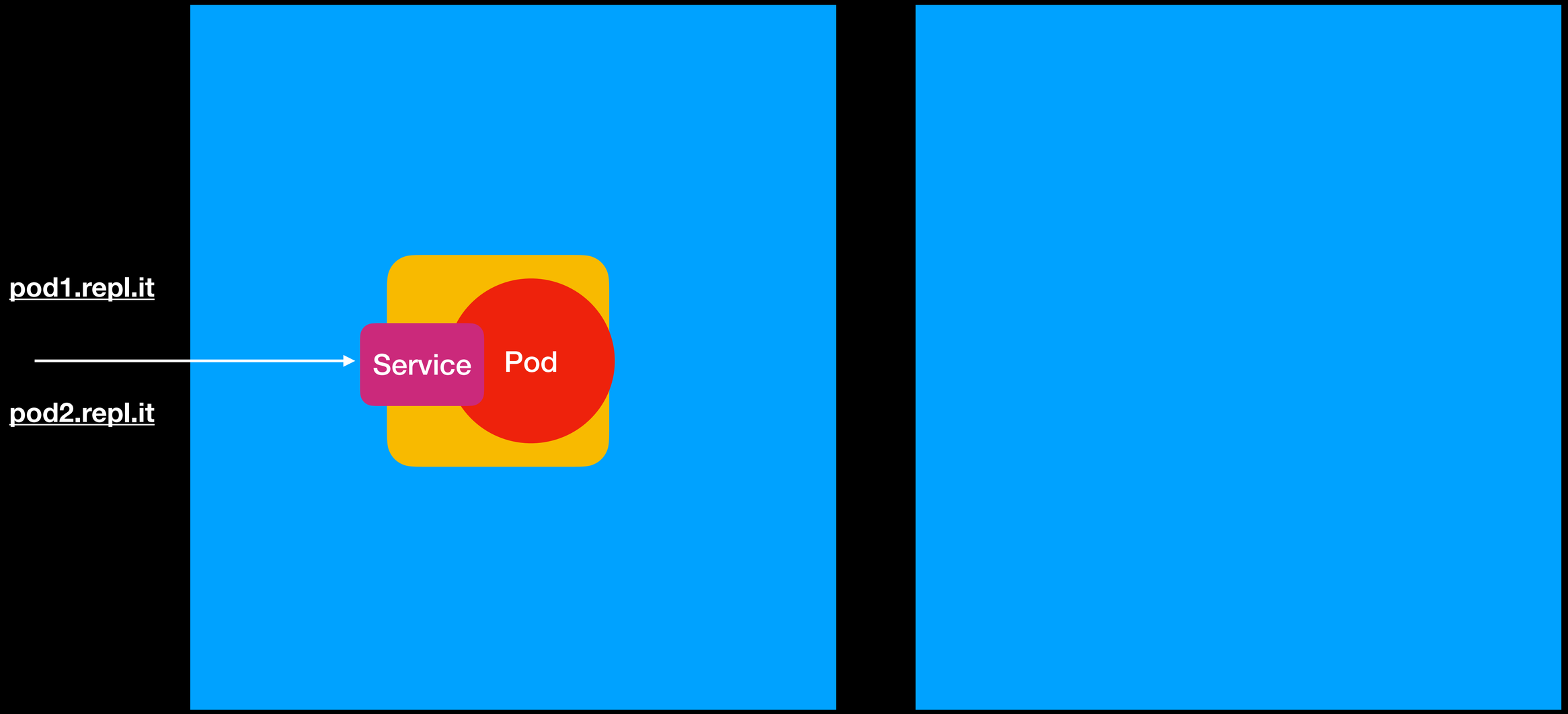




Step 2 - Attach an ingress controller to the cluster (one time)

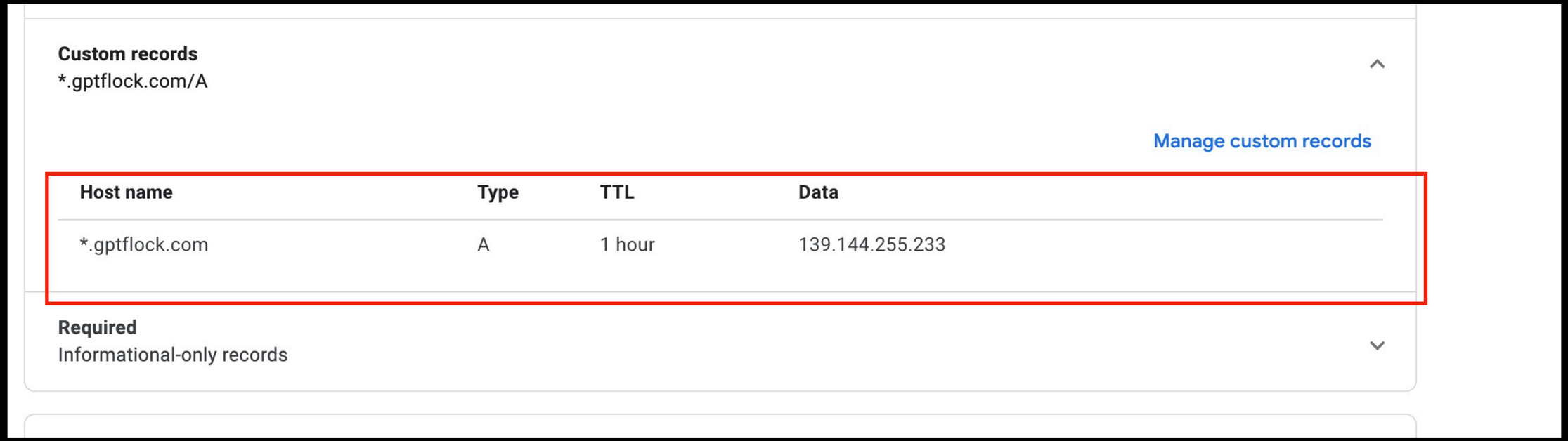


Step 3 - Point your DNS to the IP of the ingress controller



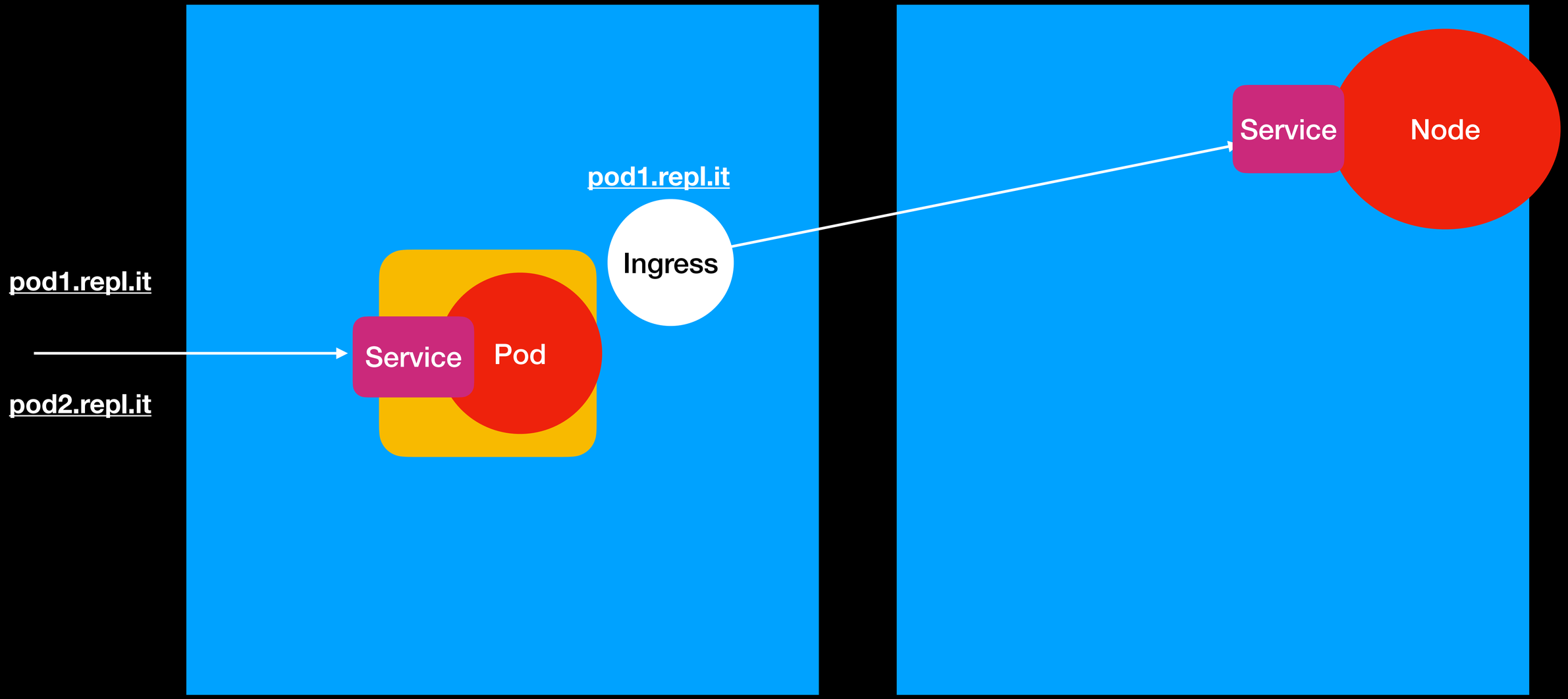
Container Orchestration - Kubernetes

Step 3 - Point your DNS to the IP of the ingress controller



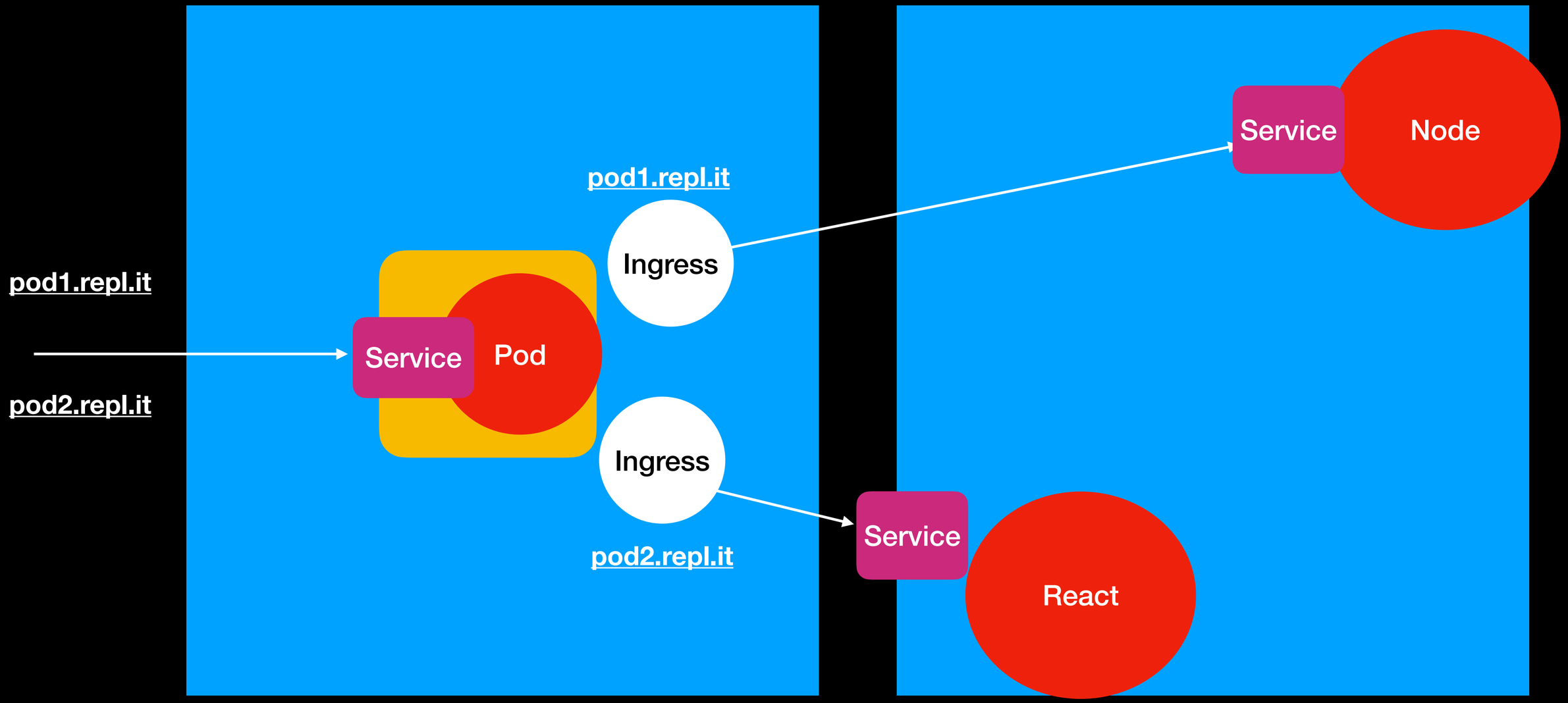
Container Orchestration - Kubernetes

Step 4 - As people start repls, start a pod, service and ingress for them



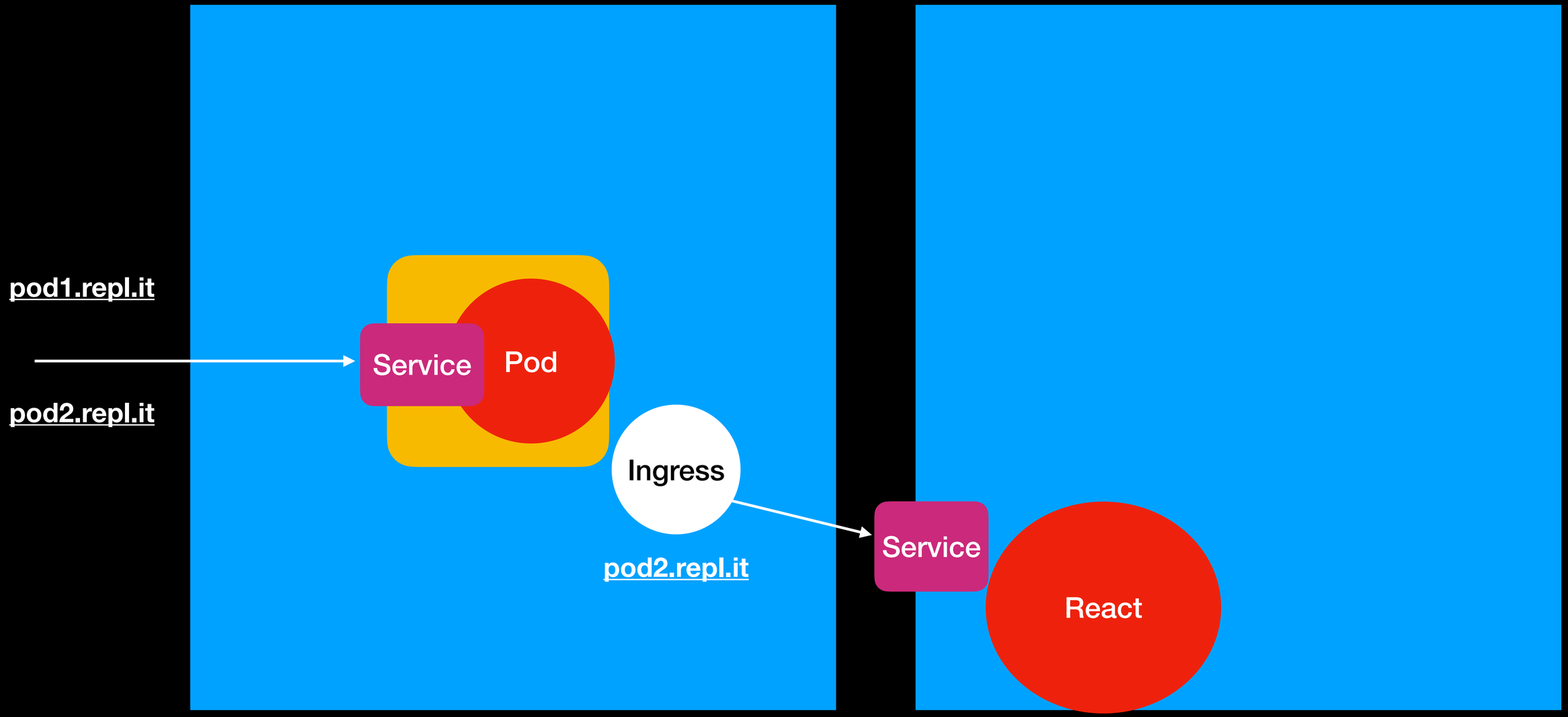
Container Orchestration - Kubernetes

Step 4 - As people start repls, start a pod, service and ingress for them



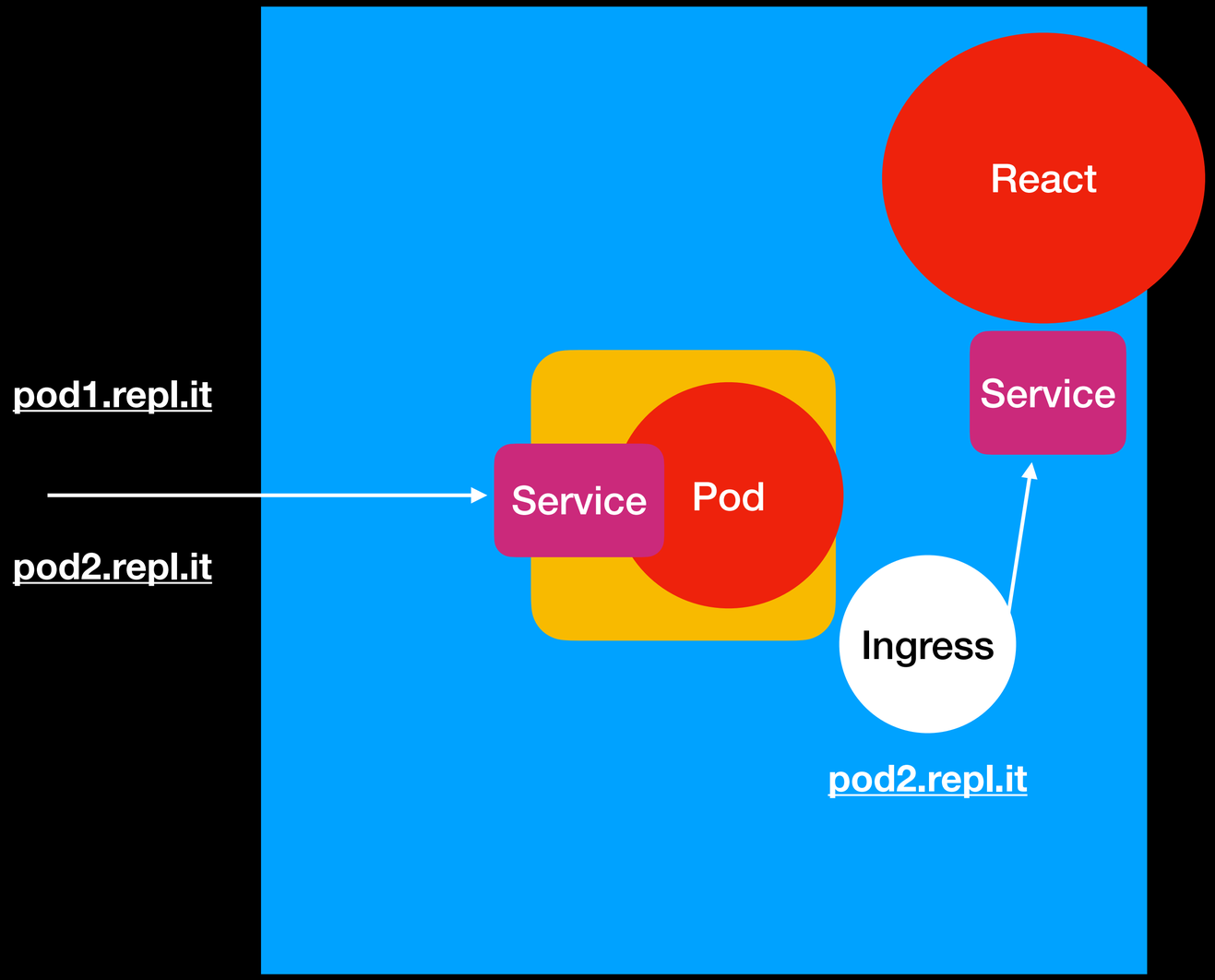
Container Orchestration - Kubernetes

Step 4 - As people leave repls, stop the respective pod, service and ingress



Container Orchestration - Kubernetes

Step 4 - Cluster will autoscale based on the policies you added



Container Orchestration - Kubernetes

You don’t NEED Kubernetes for what we’re doing today
Kubernetes gives you a bunch of things that we don’t need today (for example deployments, automatic restarts, recycling containers …

Part 2 | The good solution

Some more jargon

Basic Jargon

1. Containers (Docker)
2. Container Orchestration (Kubernetes)

Advance Jargon (not needed for this tutorial)

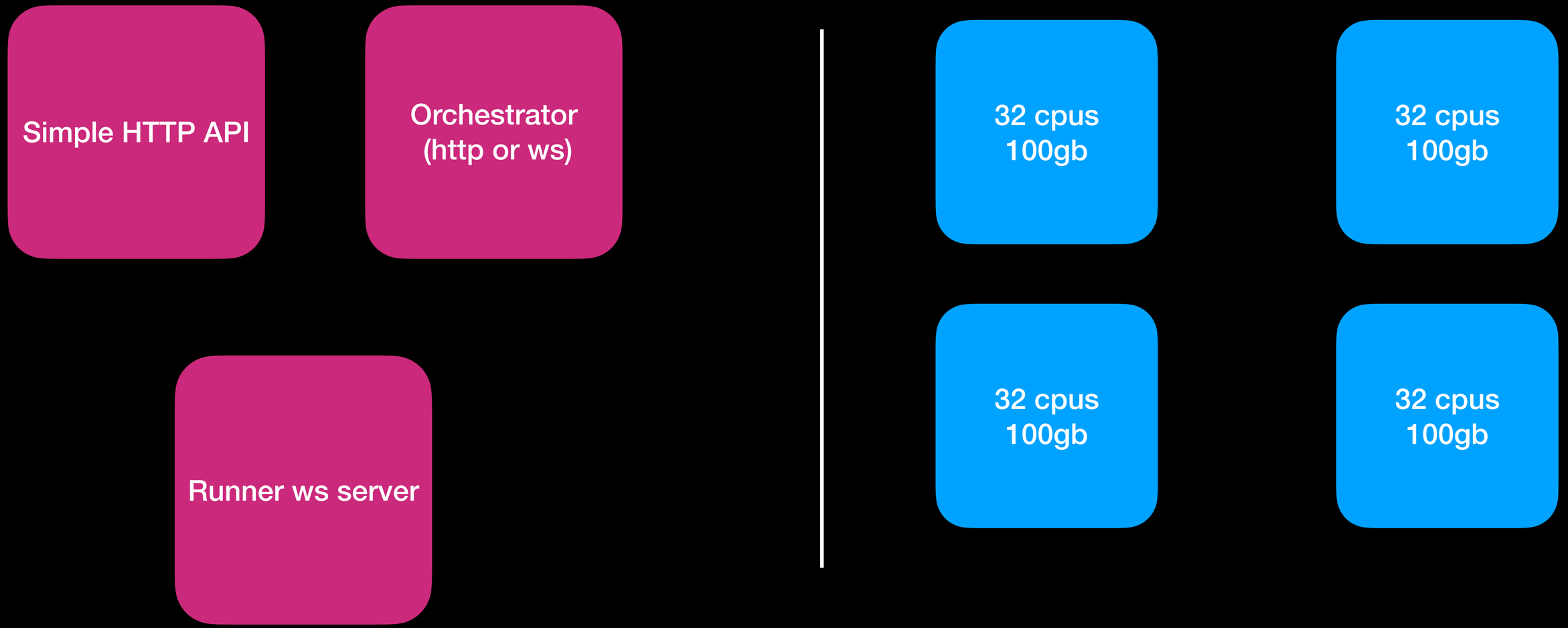
1. Reproducible builds - Nix
2. Network volumes
3. Caching dependencies

Part 2 | The good solution

My proposed solution

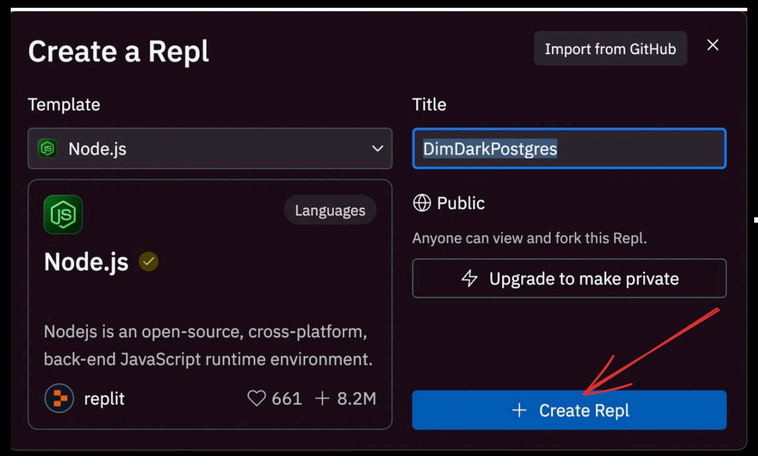
Part 2 | The good solution

We have 3 services, and a k8s cluster



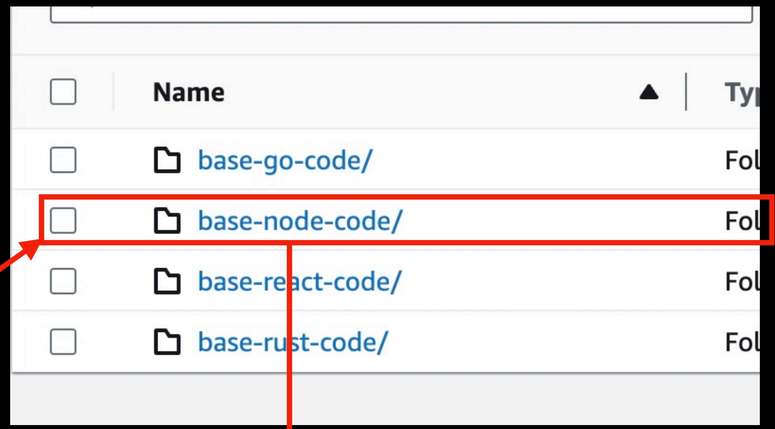
1. Simple HTTP API

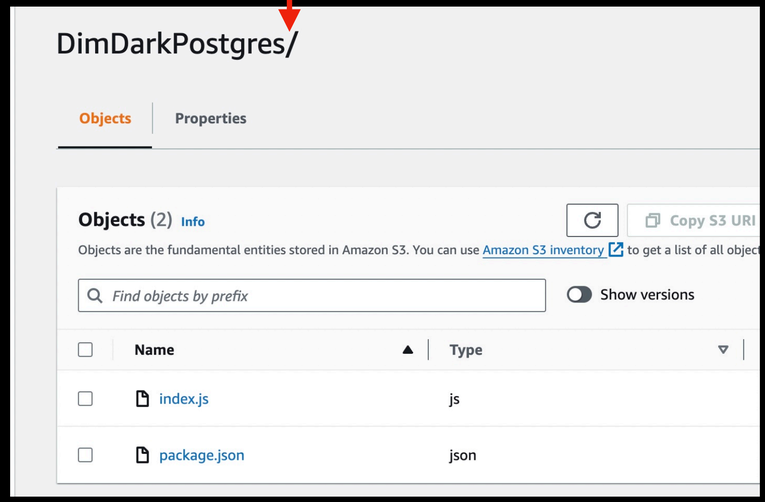
Step 1 - Initialising the repl



Simple HTTP API

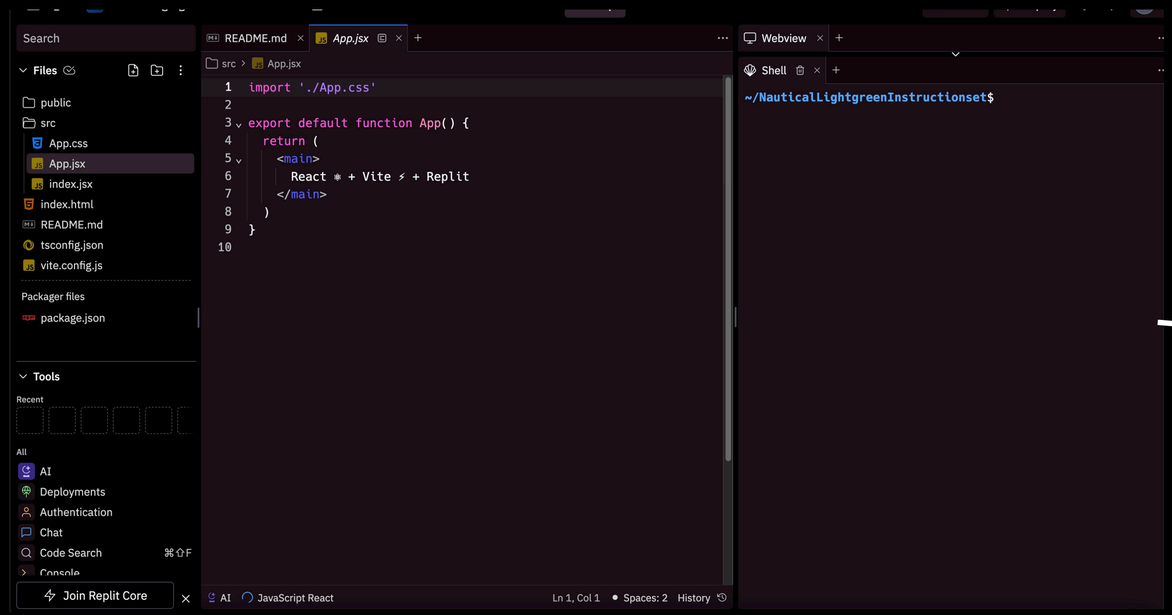
Copy over the base image to s3://images/{id}



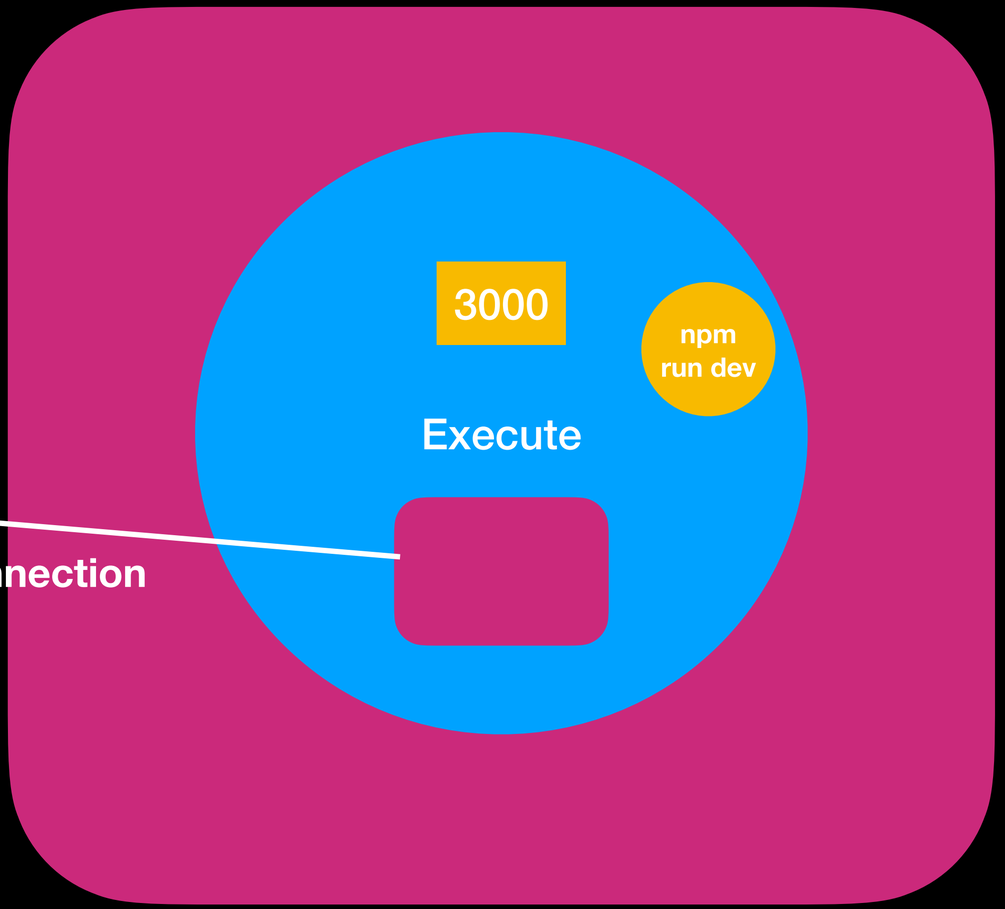


2. Runner Service

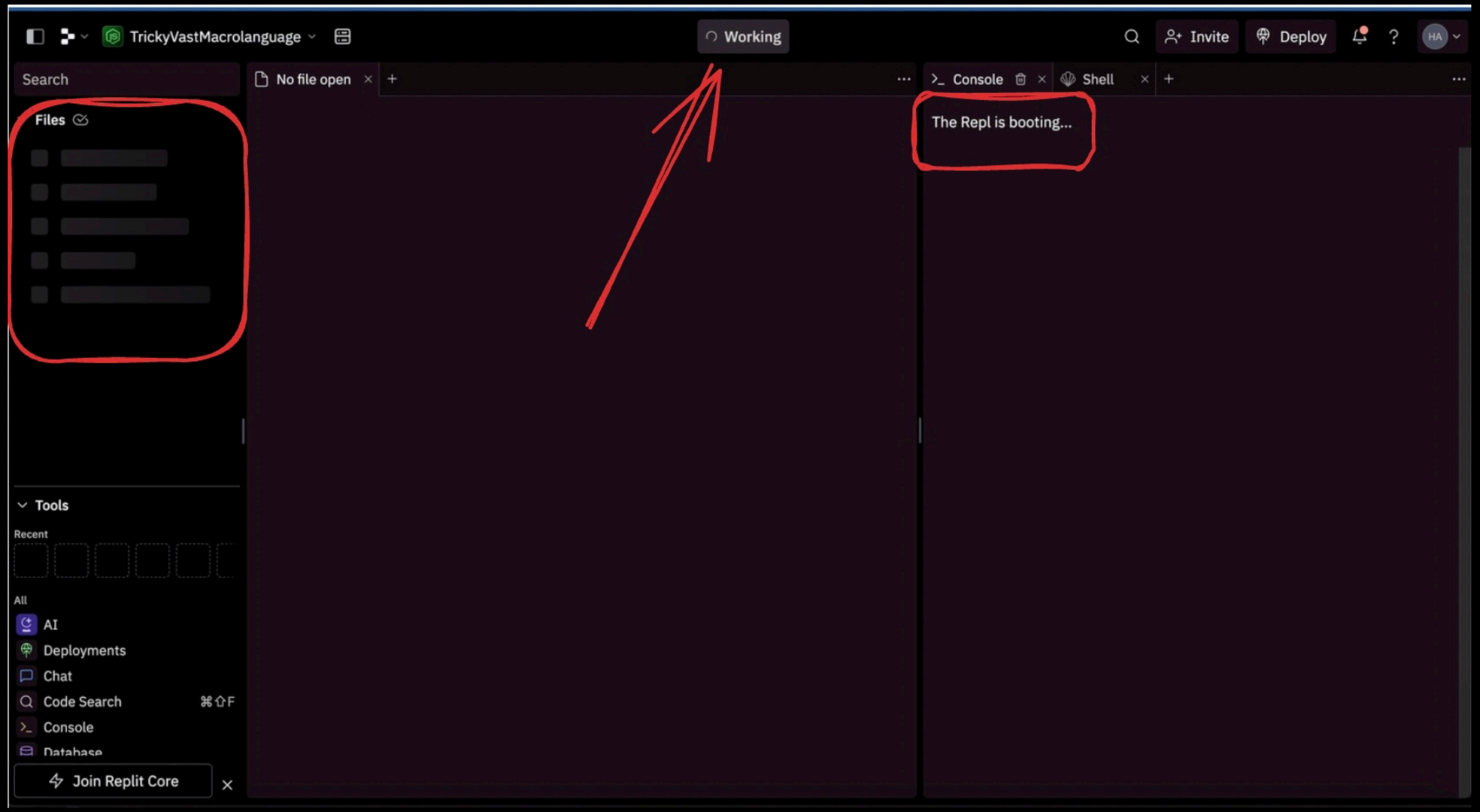
This is the same as the web socket service we built in the last section



Ws connection

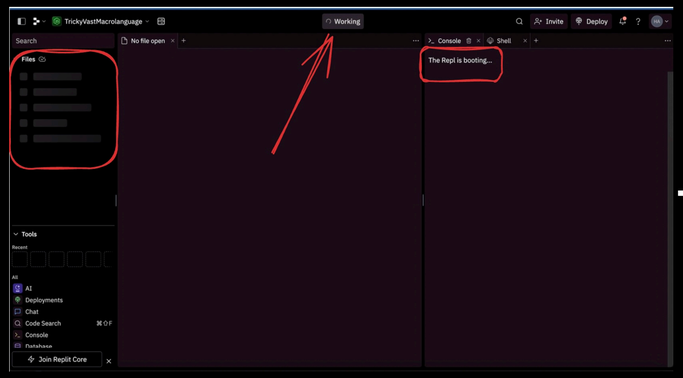


What happens after the user starts the repl?
We need to start an independent runner for them
While it starts, the user sees the loading screen



3. Orchestrator

Step 3 -Tell the orchestrator to start a pod

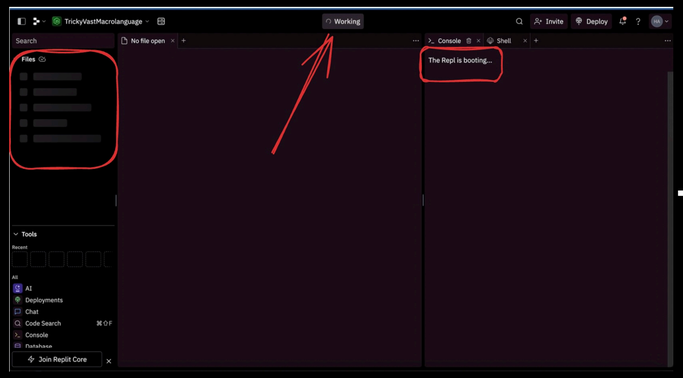


http

Orchestrator
(http or ws)

Orchestrator

Step 3 -Tell the orchestrator to start a pod



Websocket/http

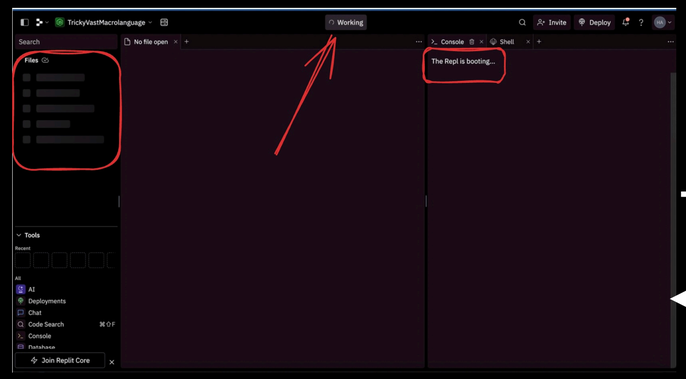
Orchestrator
(http or ws)

Runner

S3

Execution service

Step 3 -Tell the orchestrator to start a pod
Tells it to pull the code from S3



Websocket/http

runner_addr
Token

Orchestrator
(http or ws)

Runner

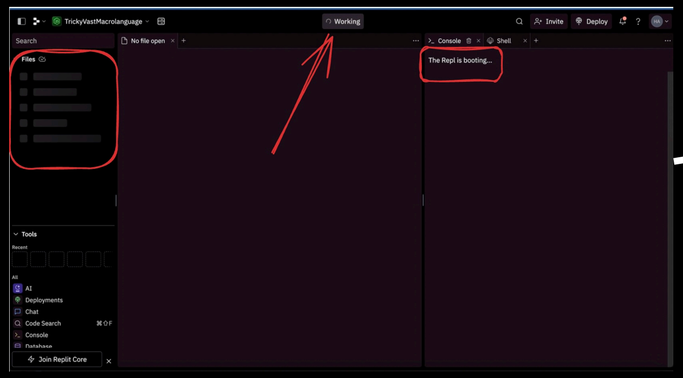
S3

Callout -

1. Caching is super helpful here
2. You can maintain a warm pool of pods that you can auto assign immediately

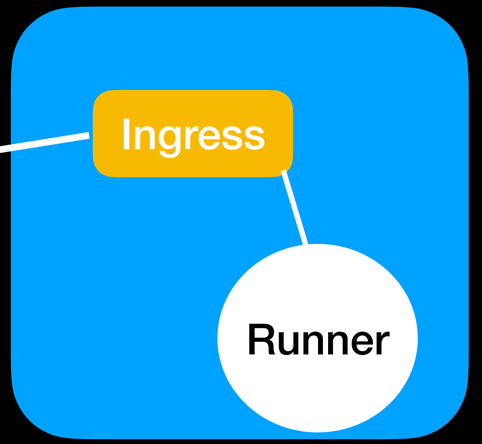
Execution service

Step 3 -Tell the orchestrator to start a pod



runner_addr
Token

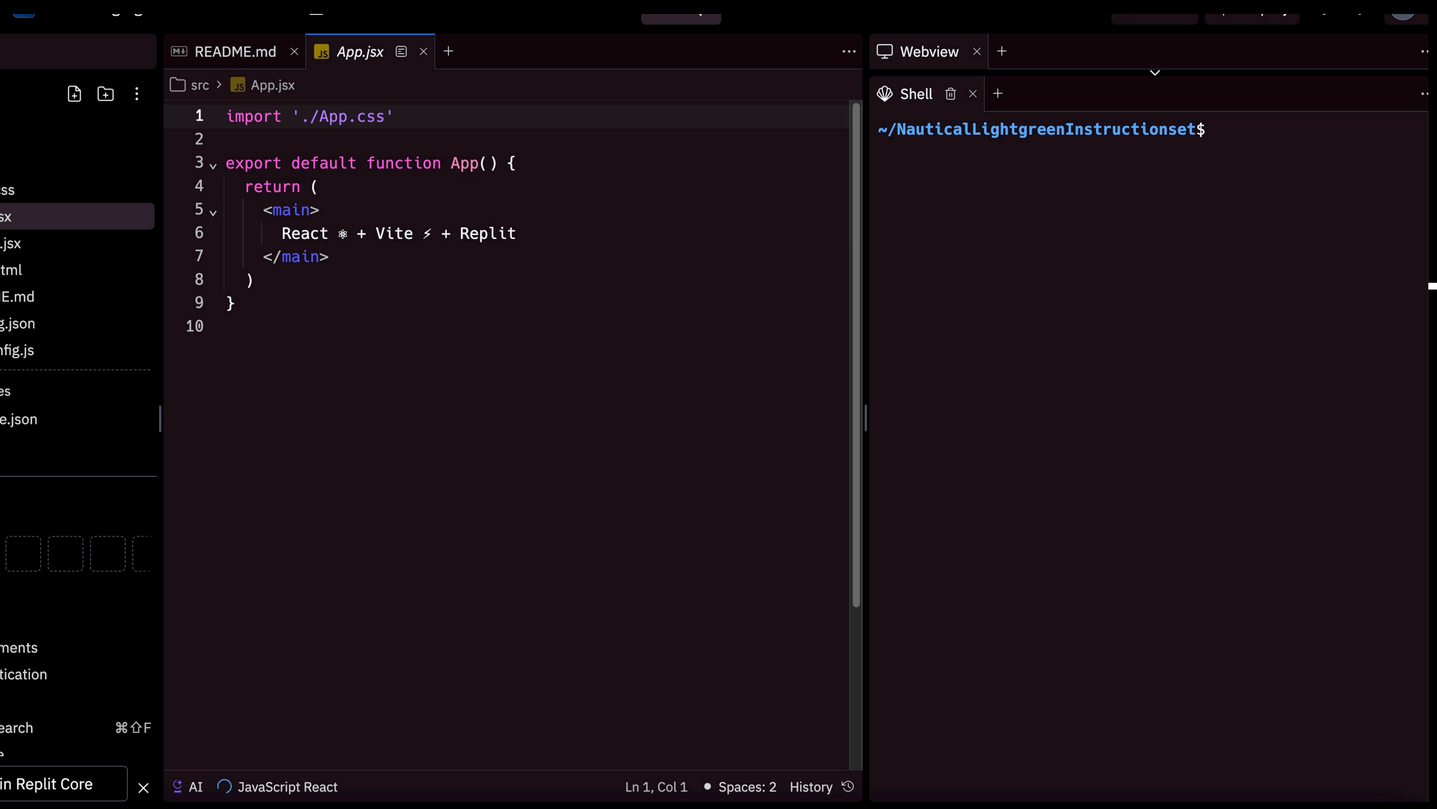
Websocket

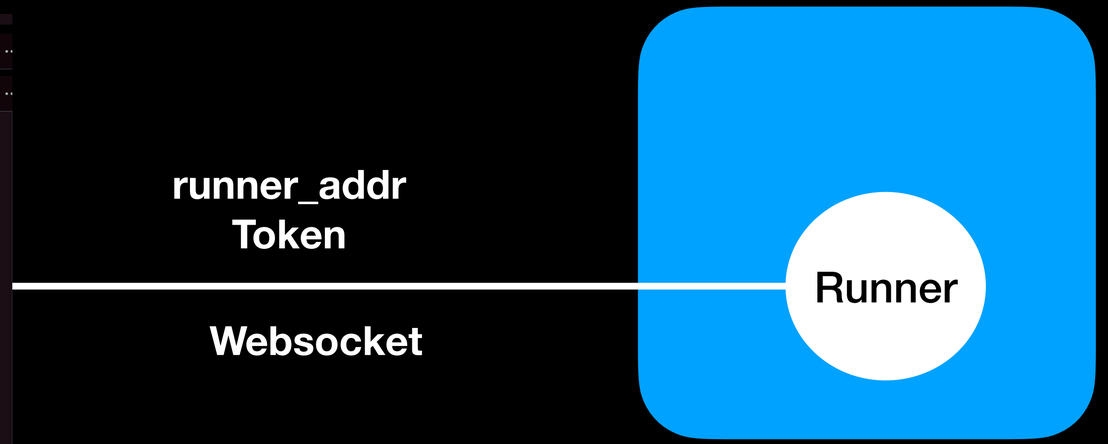


Callout -
1. Caching is super helpful here
2. You can maintain a warm pool of pods that you can auto assign immediately

Execution service

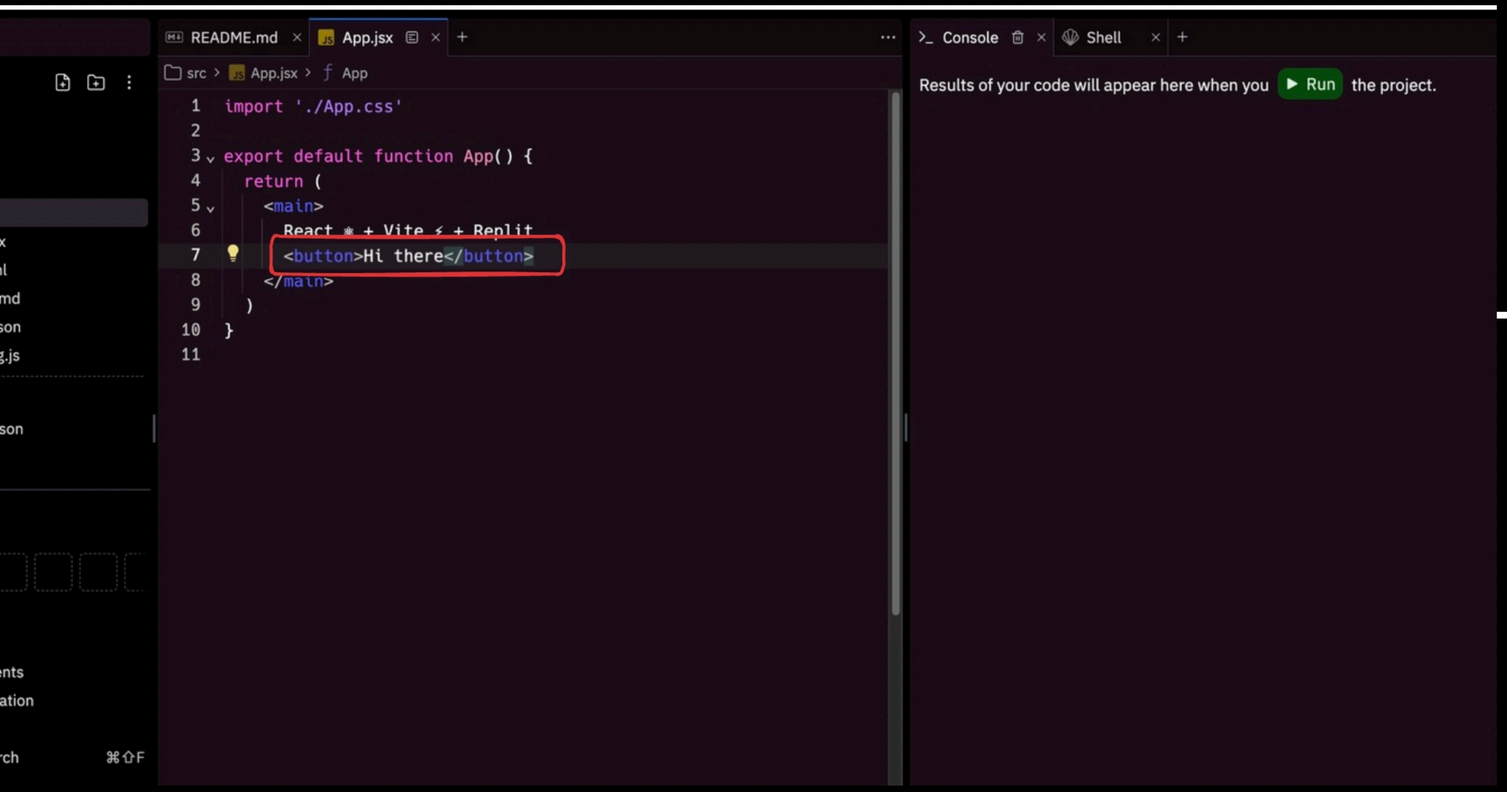
Step 4 - Let the user edit a file, send over diff over the ws layer





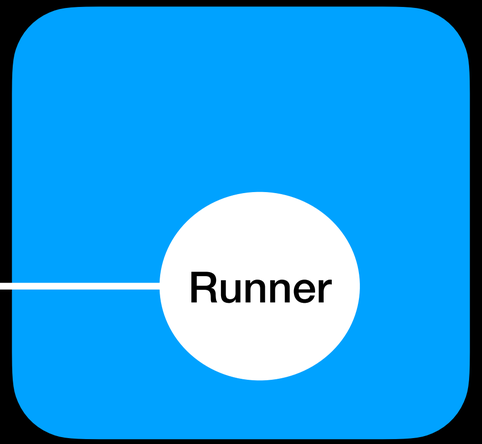
Execution service

Step 4 - Let the user edit a file, send over diff over the ws layer



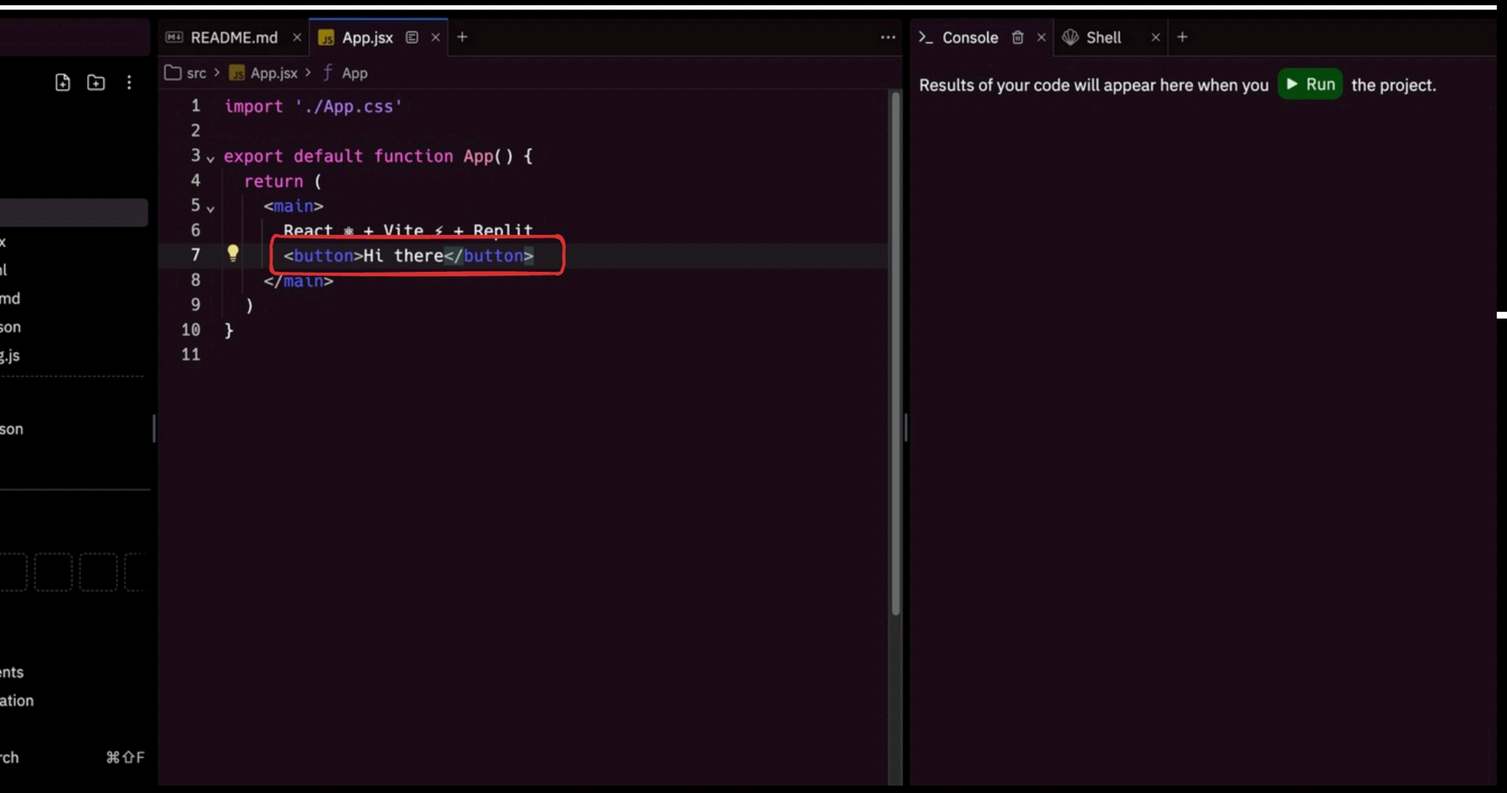
runner_addr
Token

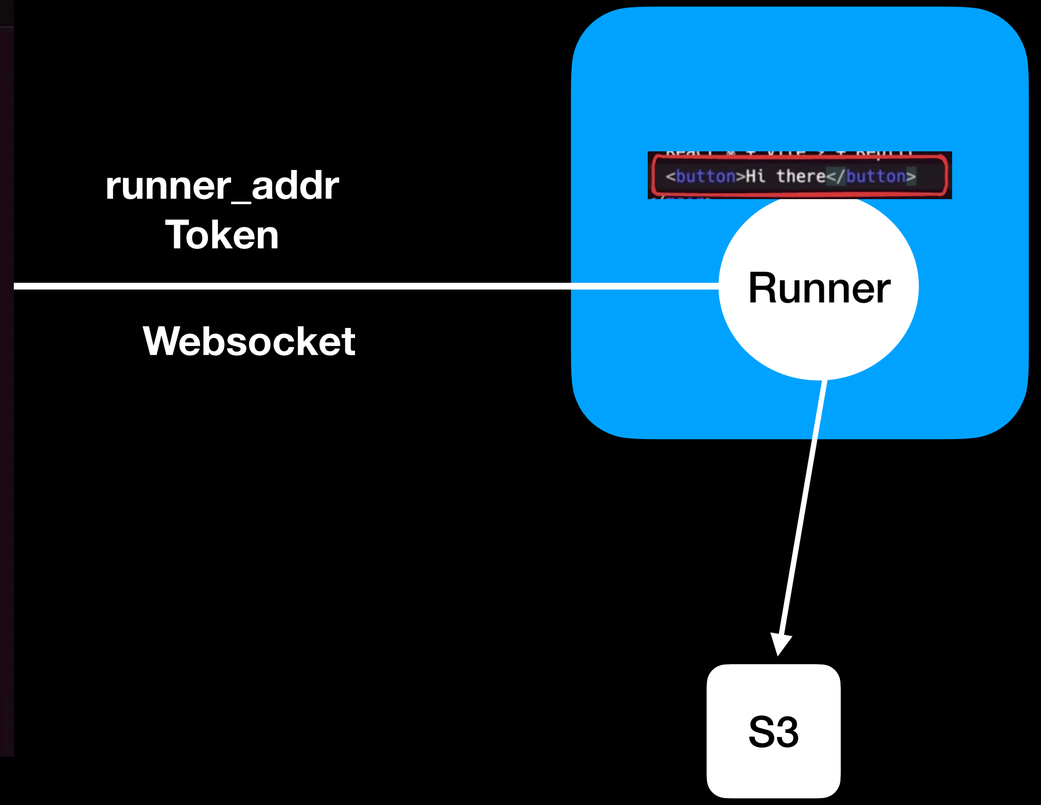
Websocket



Execution service

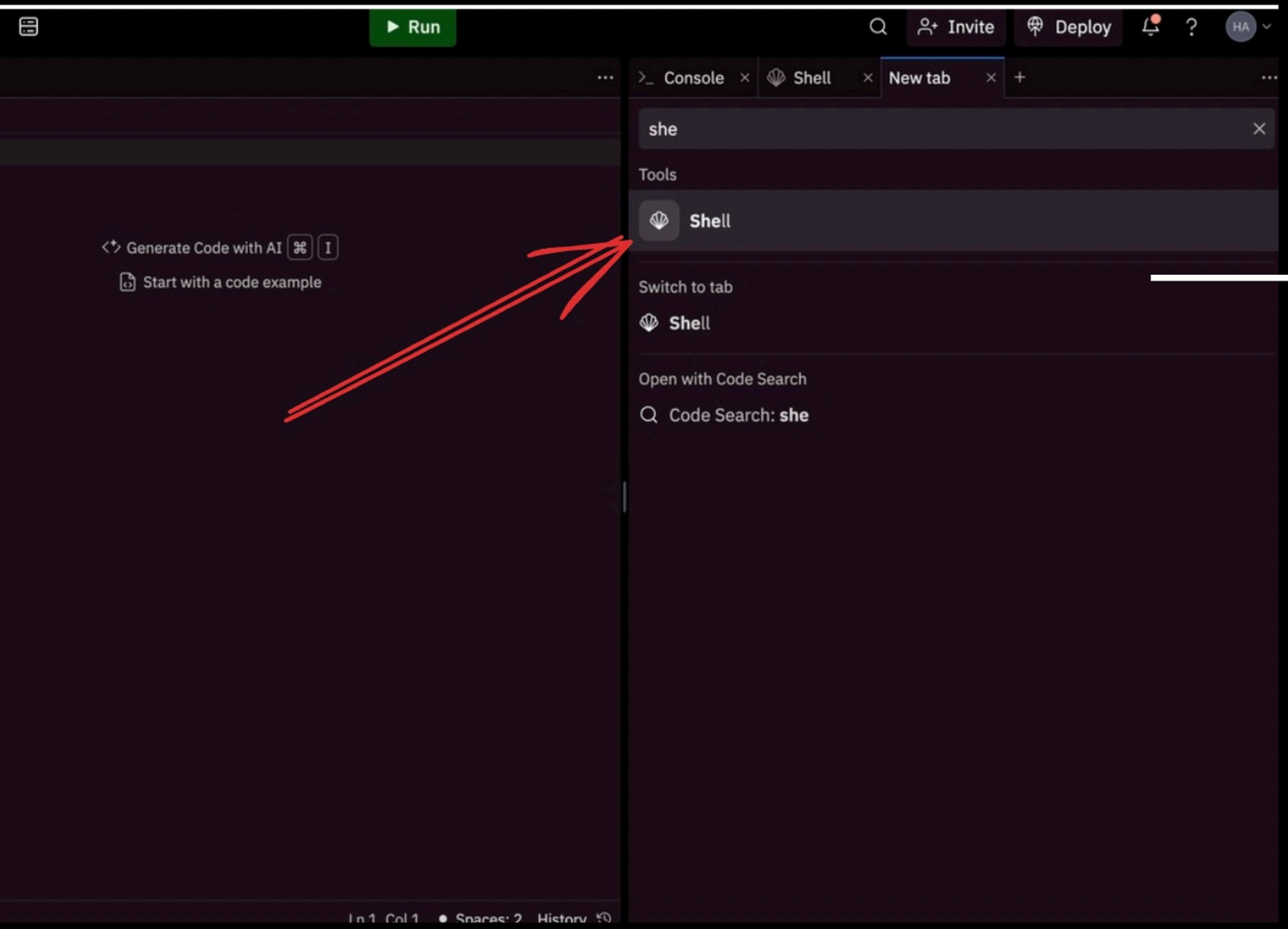
Step 4 - Let the user edit a file, send over diff over the ws layer





Execution service

Step 5 - Terminal access



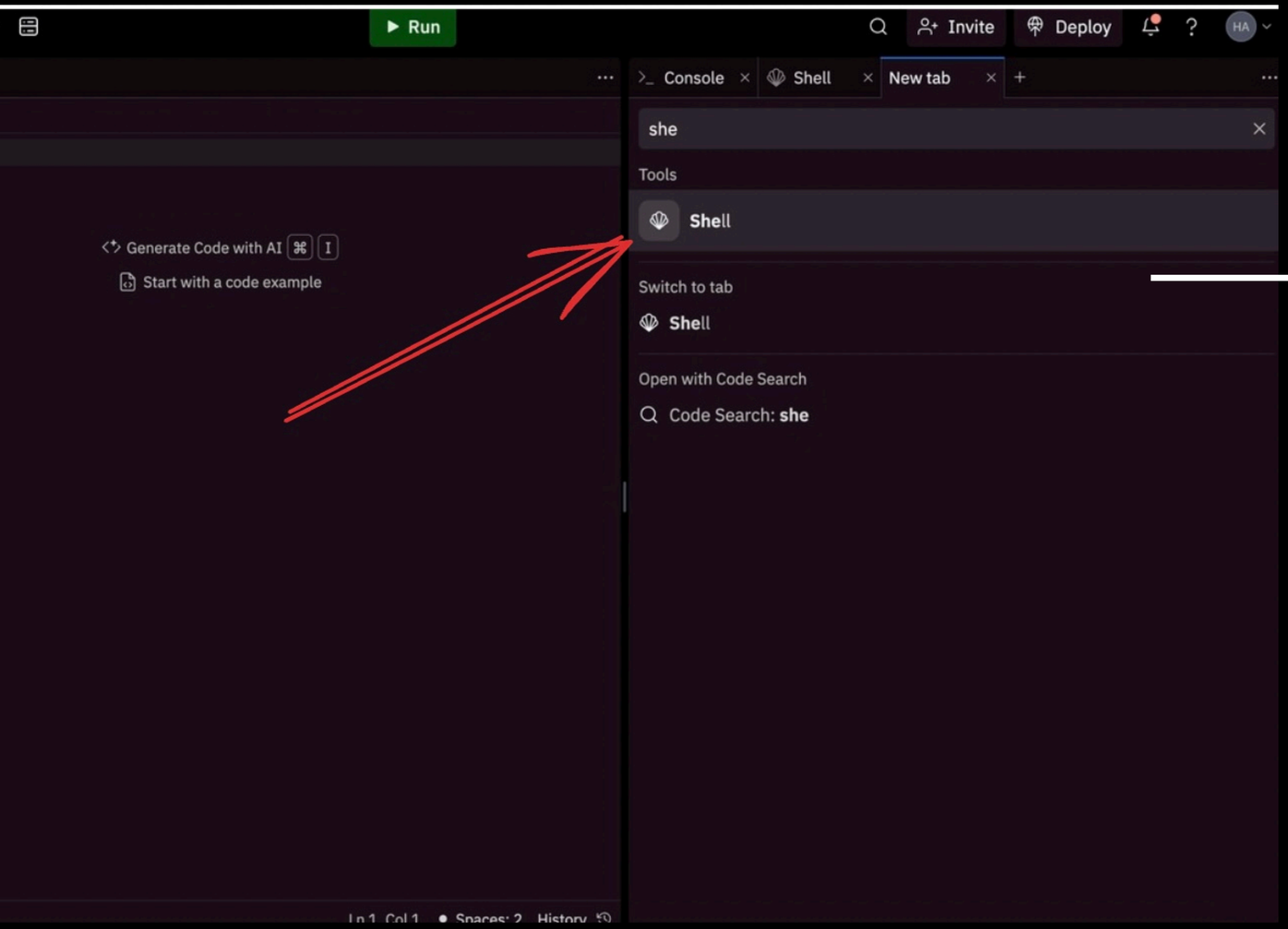
startSession

Websocket

Runner

Execution service

Step 5 - Terminal access



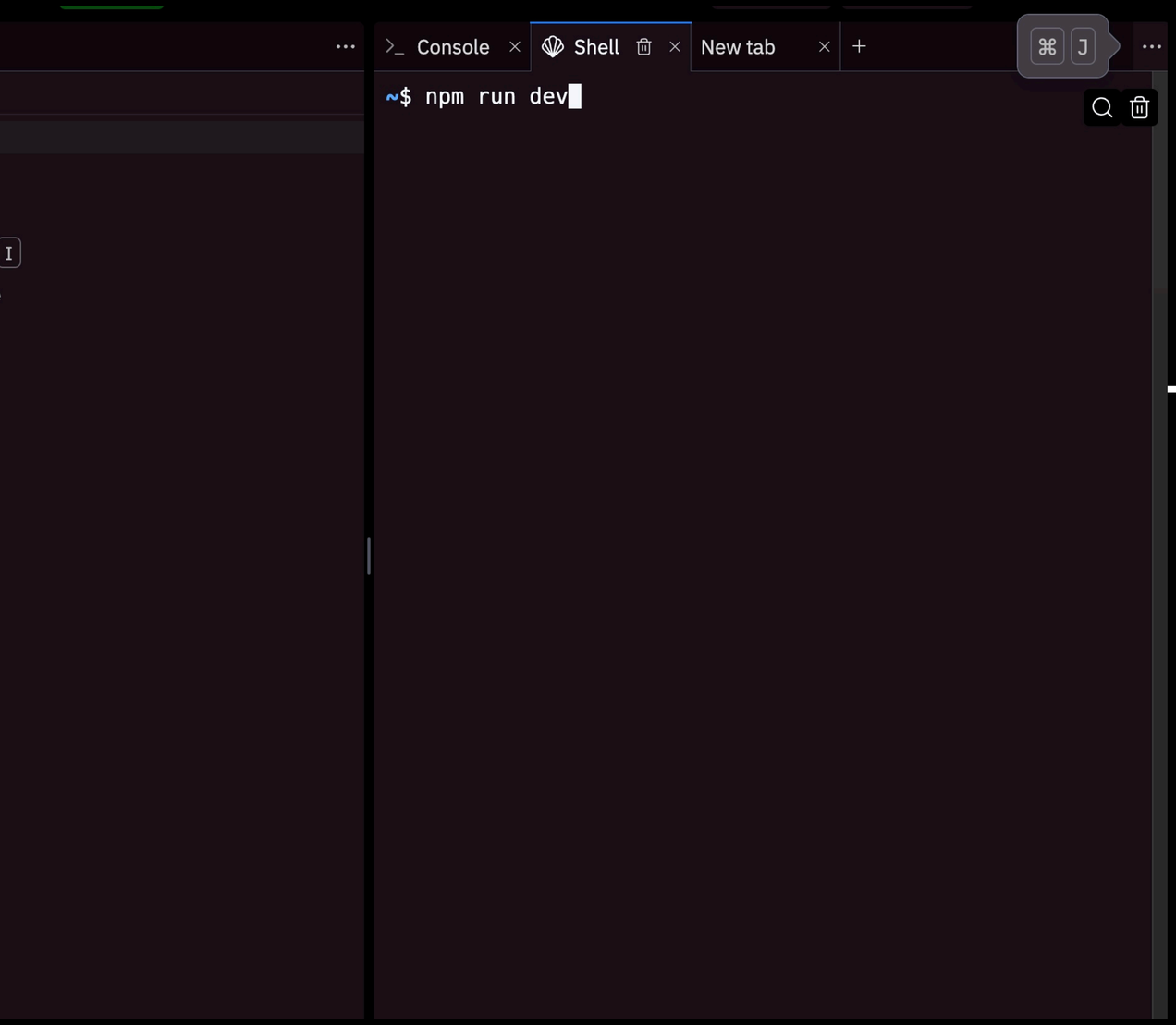
startSession

Websocket

Starts a pseudo terminal

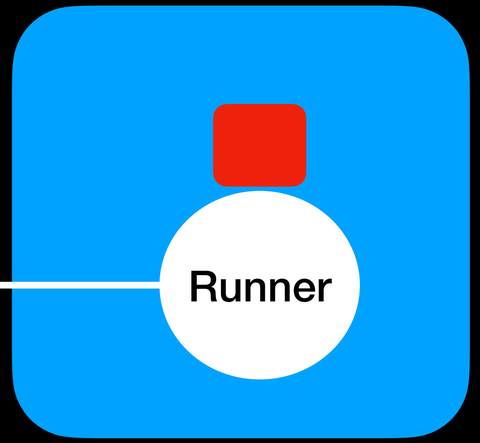
Runner

Execution service



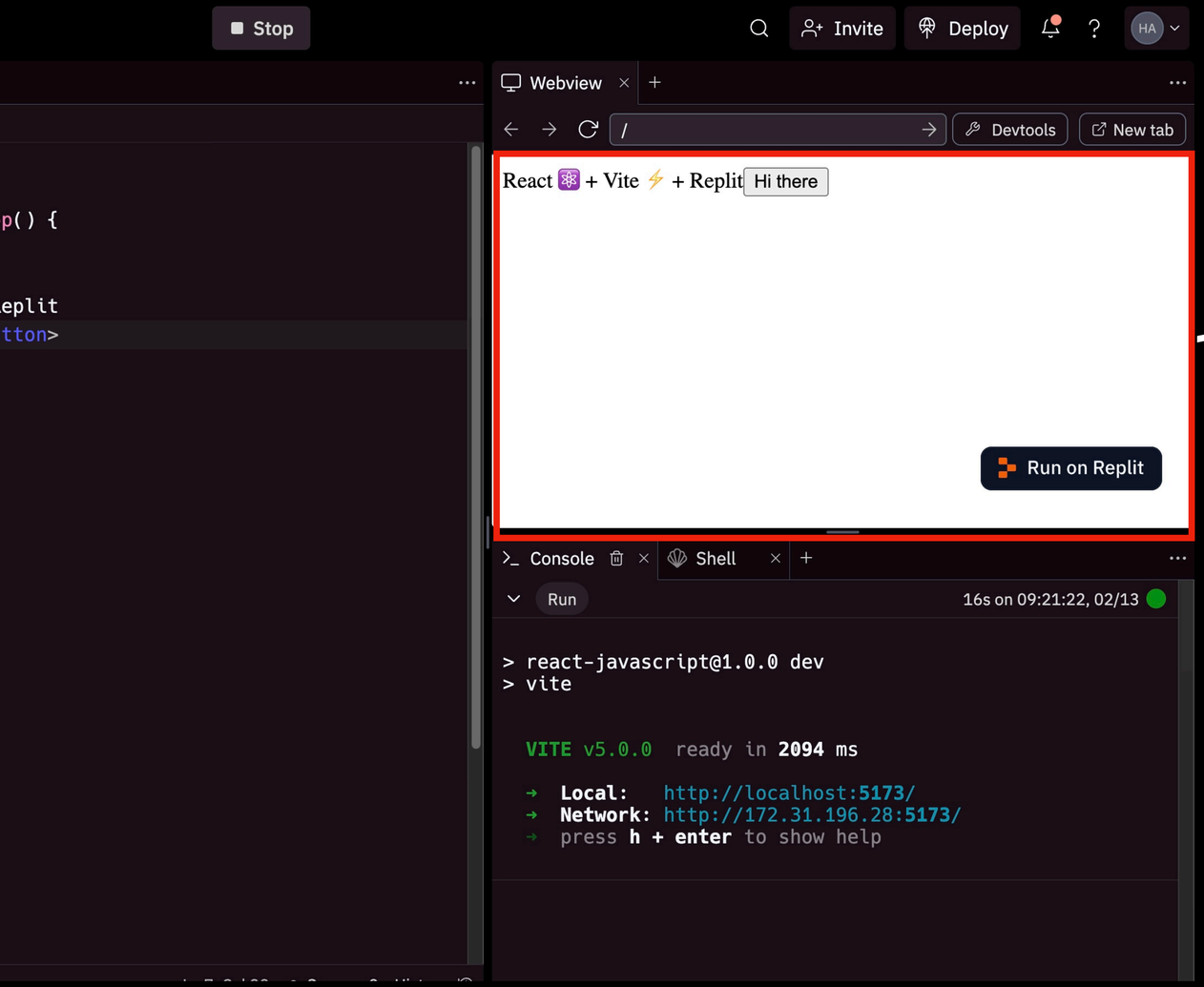
Step 5 - Terminal access

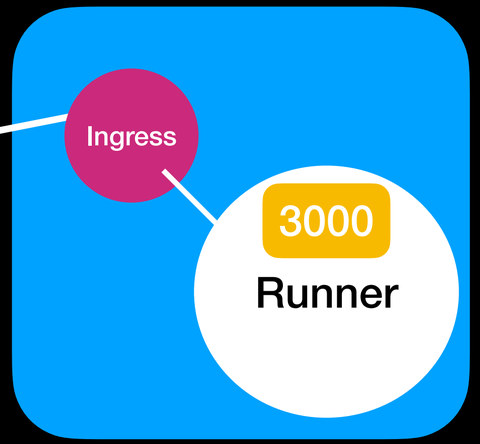
Relay keystrokes/commands



Execution service

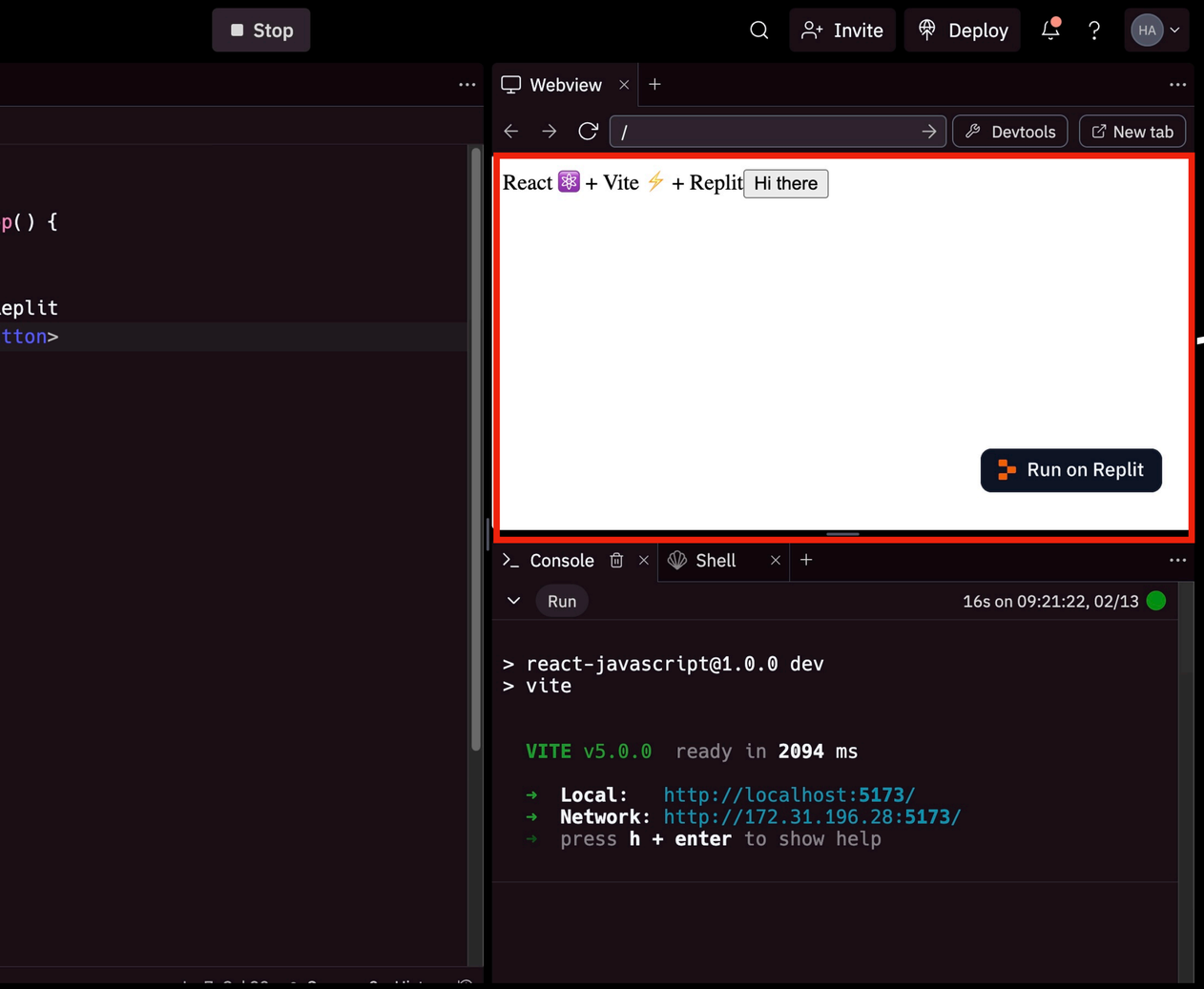
Step 6 - Accessing a process

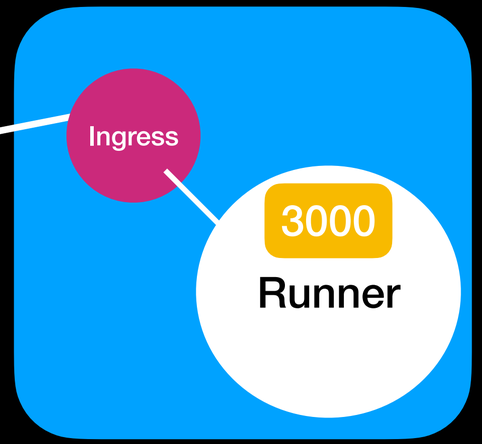




Execution service

Step 7 - Destroying the pod
If the process has 0 ws conns alive for ~5 minutes, it can kill itself.

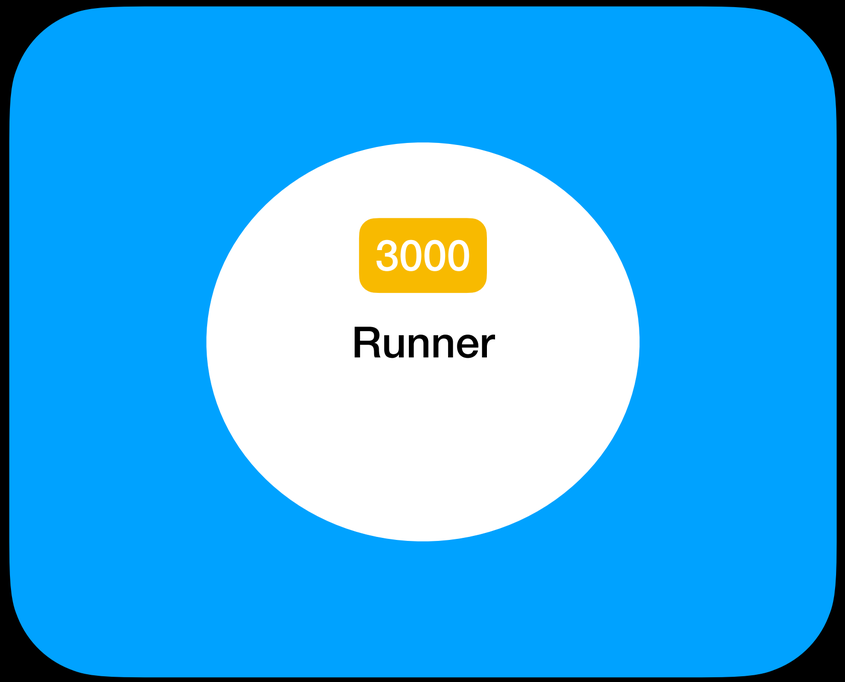




Execution service

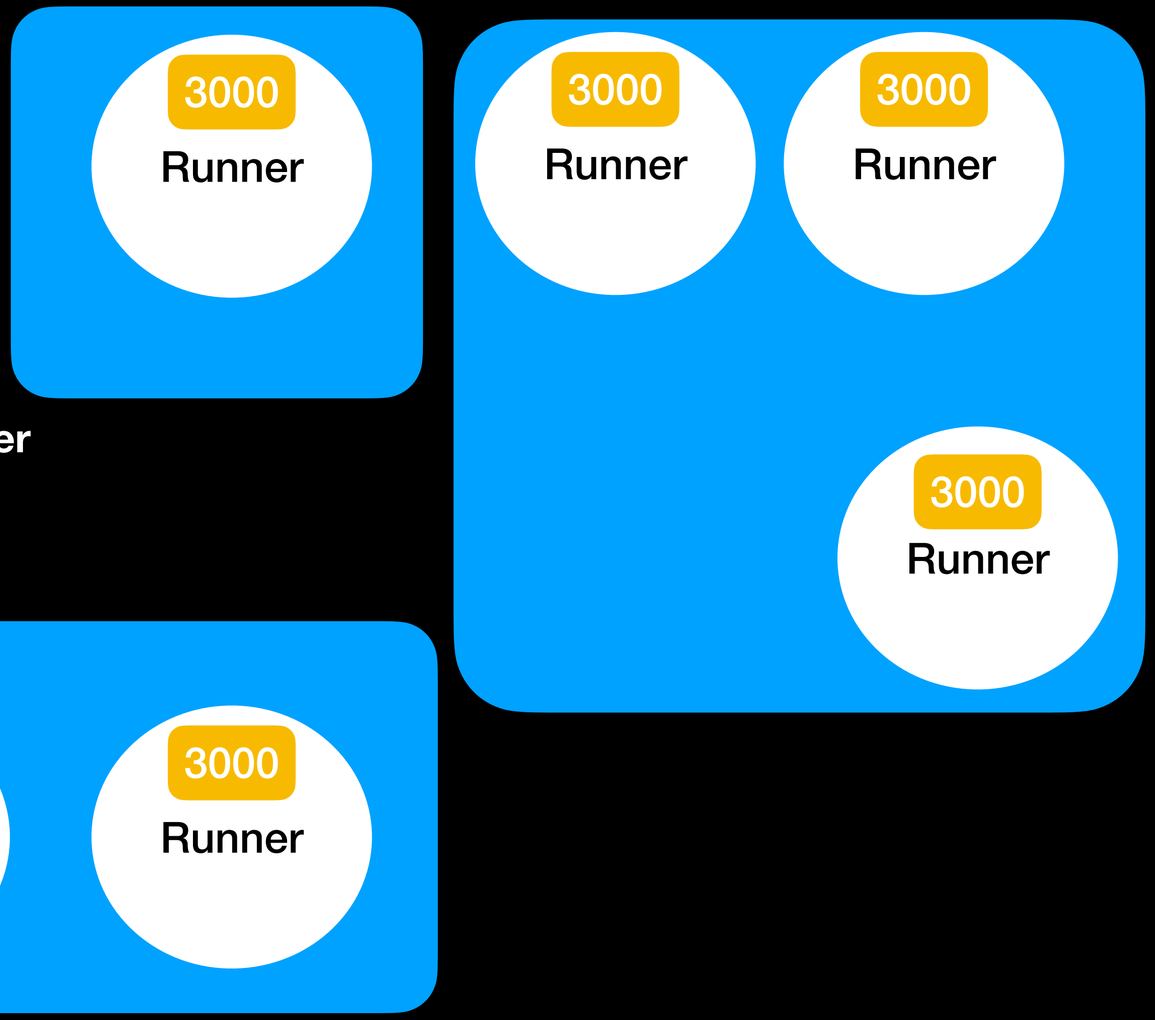
Why is this secure?

Remote code runs in a pod
Pod has restricted permissions
Pod has resource limits (only 2CPUs for example)
Pod has it’s own network (no port conflict)



Execution service

Why is this scalable?



Starting a new runner is as simple as starting a pod
As the CPU usage goes up and more pods get added,
All you have to do is increase the number of nodes in your cluster

Few callouts that might feel like good to haves

1. User can inspect your ws codebase (not good). They can stop this process as well.
   Fixing this might involve starting a parent process which takes inputs from the user
   and forwards it to the container through a socket
2. User can still reboot your machine (can do the same on replit)

Few advance things that repl.it does

1. Nix for reproducible builds
2. Network mounts to allow users to get 250Gb space on each repl
3. repl.it doesn’t let the user directly connect to the pod, but most probably relays information via a relay ws server