Breaking down the monolith with containers

aws

Speaker

Sanchit Jain

Analytics Practice Lead - AWS at Quantiphi
AWS Hero & AWS Ambassador

FOLLOW ME

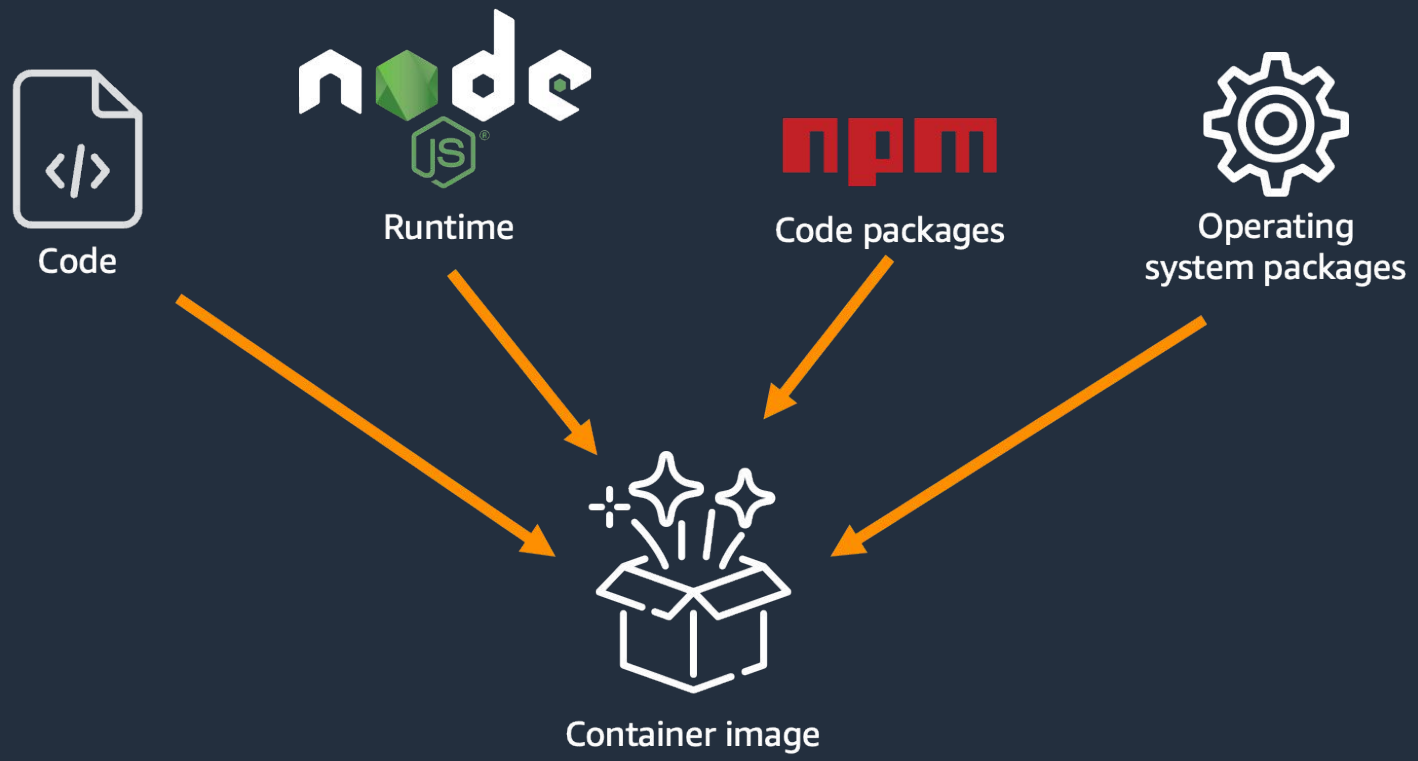
Agenda

- Why containers?
- Why decouple monoliths into microservices?
- Popular decoupling patterns with containers
- Practical decoupling tips
- A (brief) overview of our container services

Why containers?

Applications aren’t just code, they have dependencies

Containers turn applications into one deployable artifact

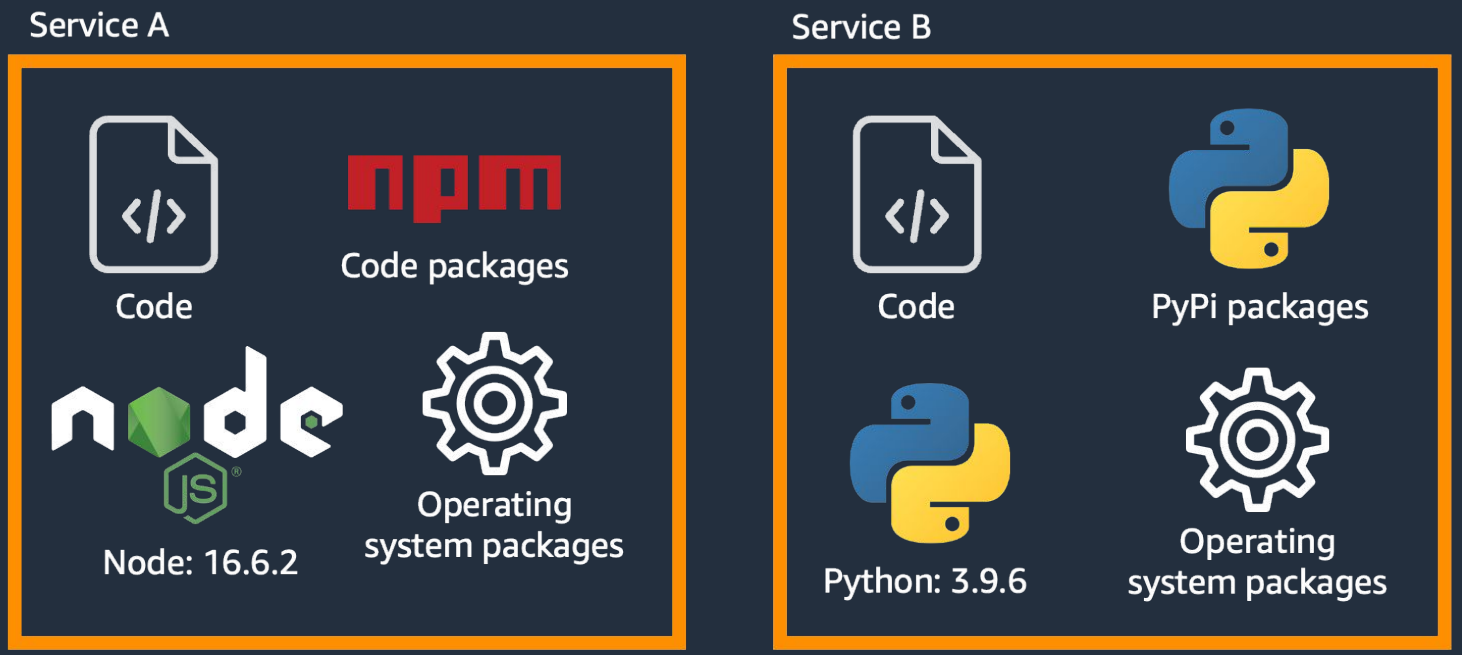


Build
Gather the app and its dependencies. Create an immutable container image.

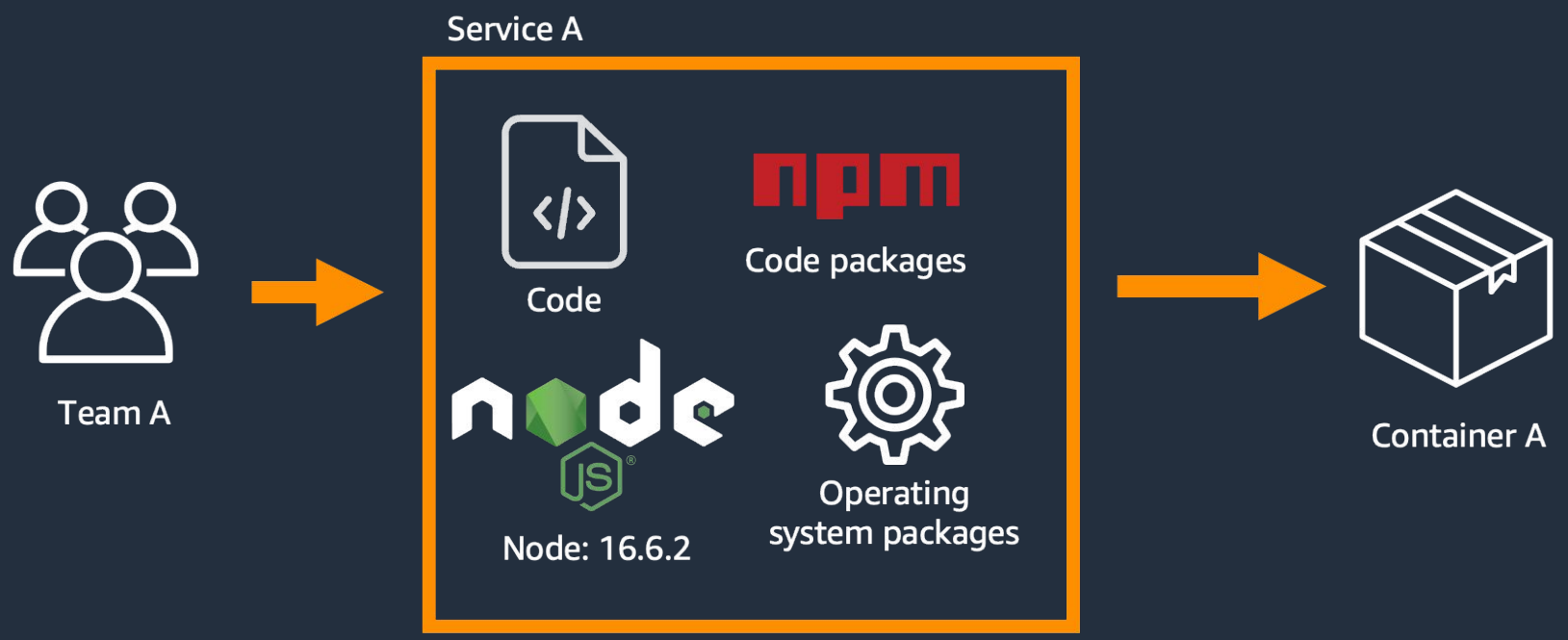
Push
Store the container image in a registry so it can be downloaded to compute

Run
Download image to compute, unpack it, and run it in an isolated environment

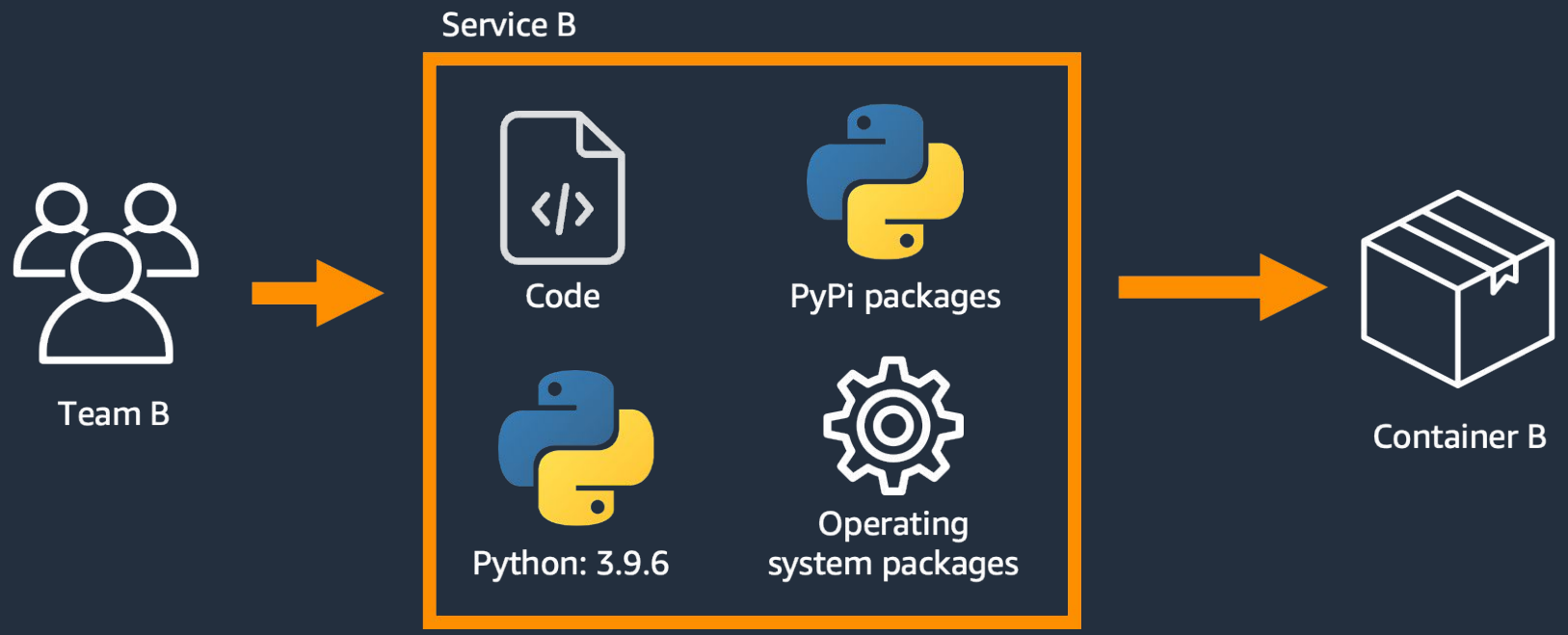
Breaking a monolith is scary because more services mean more dependencies



Containers make dependencies a decentralized job



Containers make dependencies a decentralized job



Why decouple monoliths into microservices?

Decoupling your services = decoupling your teams

API = Contract
w/SLAs,
versioning, etc.

“Smaller teams working on smaller codebases tend to be more productive.”

“Can we make a change to a microservice and deploy it without having to deploy any other?”

Popular decoupling patterns with containers

Listener Rule

Up to 100 rules

Match on host

Hostname == mycompany.com
Hostname == api.mycompany.com

Match on path

Path == /api/users
Path == /api/orders

Match on query string

?utm_source==bot

Match on header

Version == 1.0.0
User-Agent == mobile

Practical decoupling tips

Practical decoupling: from monolith to micro

Functioning monolith

Trying to break things up too fast is a recipe for disaster

Kind of works?

Semi-functioning microservice

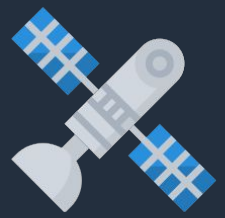
Buggy service

Boom!

Broken implementation

This part works great!

Decouple gradually, leave the central monolith for a while

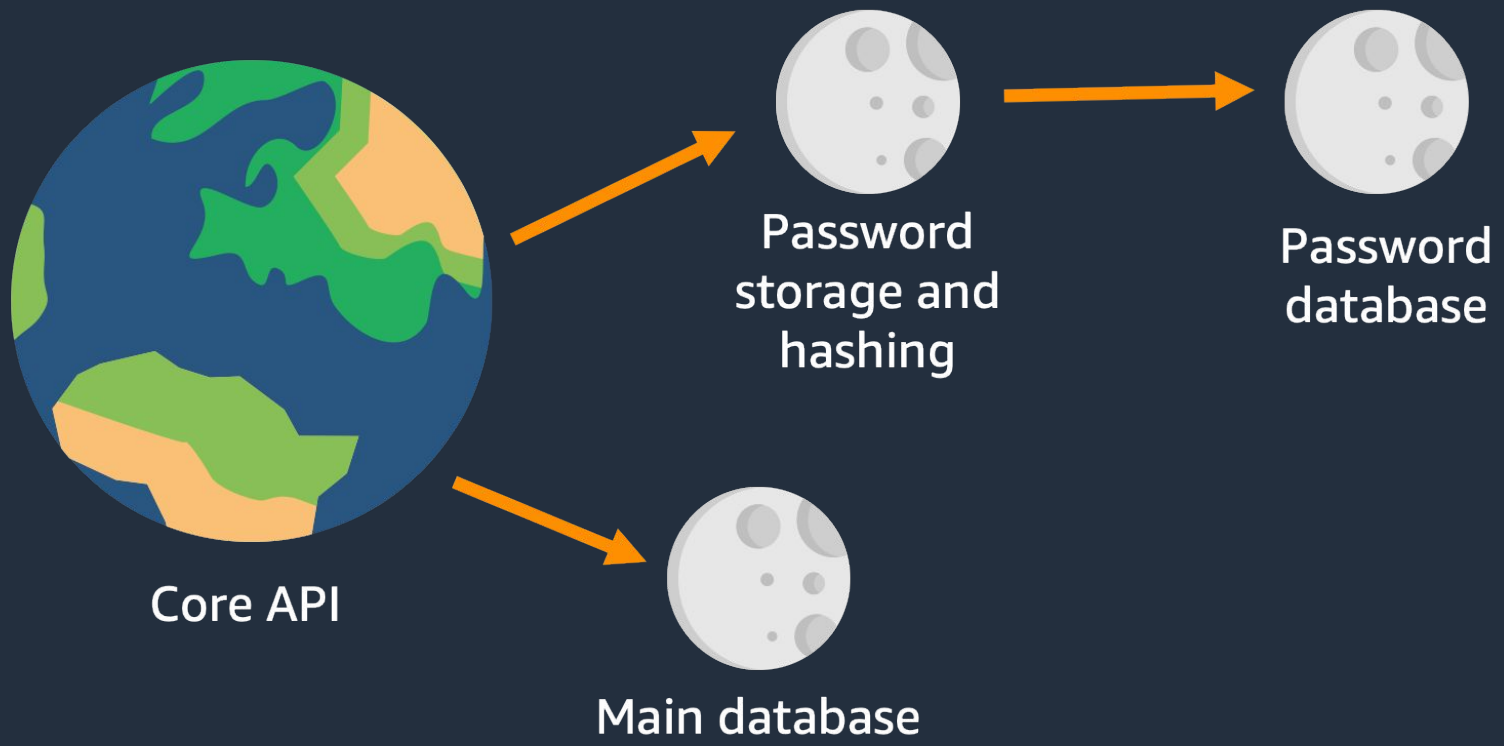


Efficient microservice

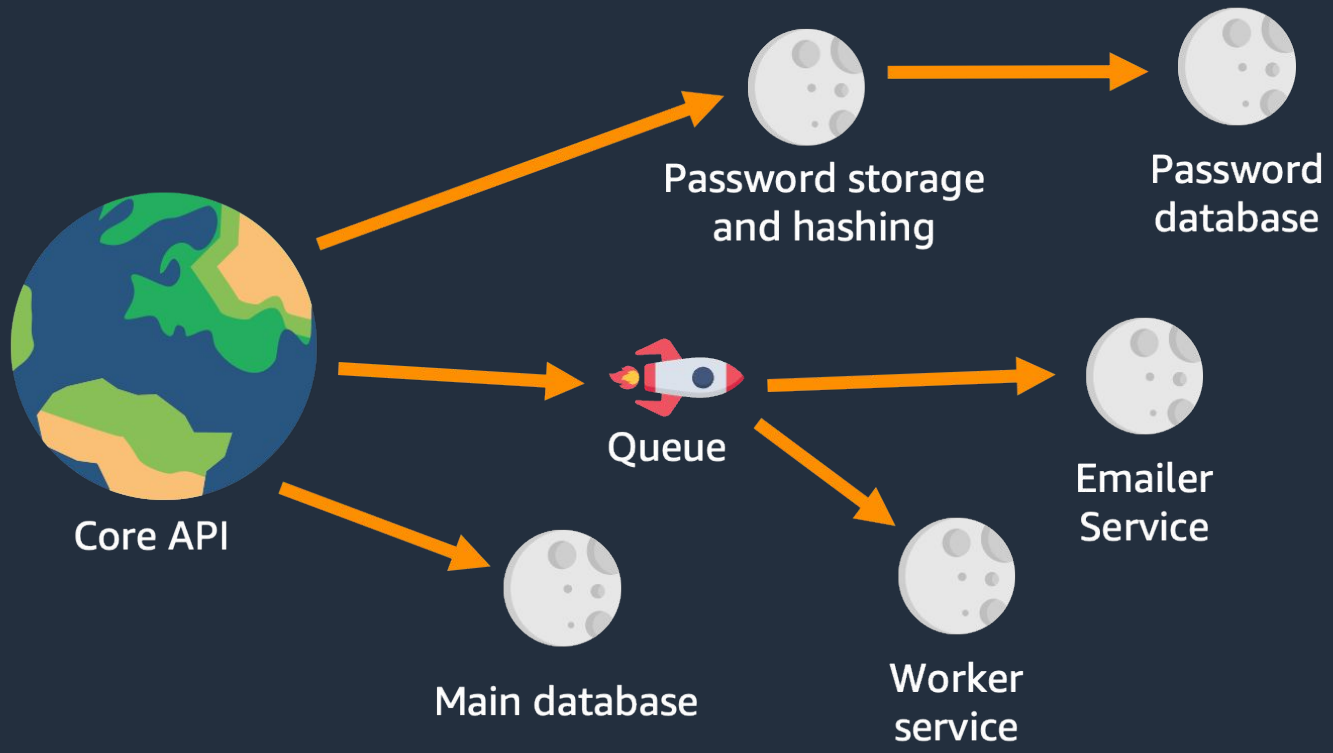
Functioning monolith

Well functioning service

Some practical places to start: User signup



Some practical places to start: User signup



Where to start?

When deciding what parts of your app to spin out of the monolith you should look for transactions that:

- Can be made asynchronous
  - vs. a monolith that is mainly synchronous
- Have well above average response times
  - For example, just that transaction could be re-written in Rust/Go vs. Python
- Have different resource requirements or scaling needs
  - For example, just that transaction of the app could benefit from expensive GPUs

Overview of AWS Container Services

Operating containers at scale is challenging

Security

Do we have vulnerabilities on our hosts?

Maintenance

How are we handling ongoing AMI management, logging, & monitoring?

Capacity

Is the size of our cluster properly sized and can we scale as-needed?

Cost

Are we being efficient with our spend?

Focus

Do we spend more time on our infrastructure than our applications?

Choosing your container environment

Amazon ECS

Powerful simplicity

- Fully managed containers orchestration
- Opinionated solution for containers
- Reduced time to build and deploy
- Fewer decisions needed

Amazon EKS

Open flexibility

- If you are invested in Kubernetes
- Vibrant ecosystem and community
- Consistent open-source APIs
- Easier to run K8s resiliently and at-scale

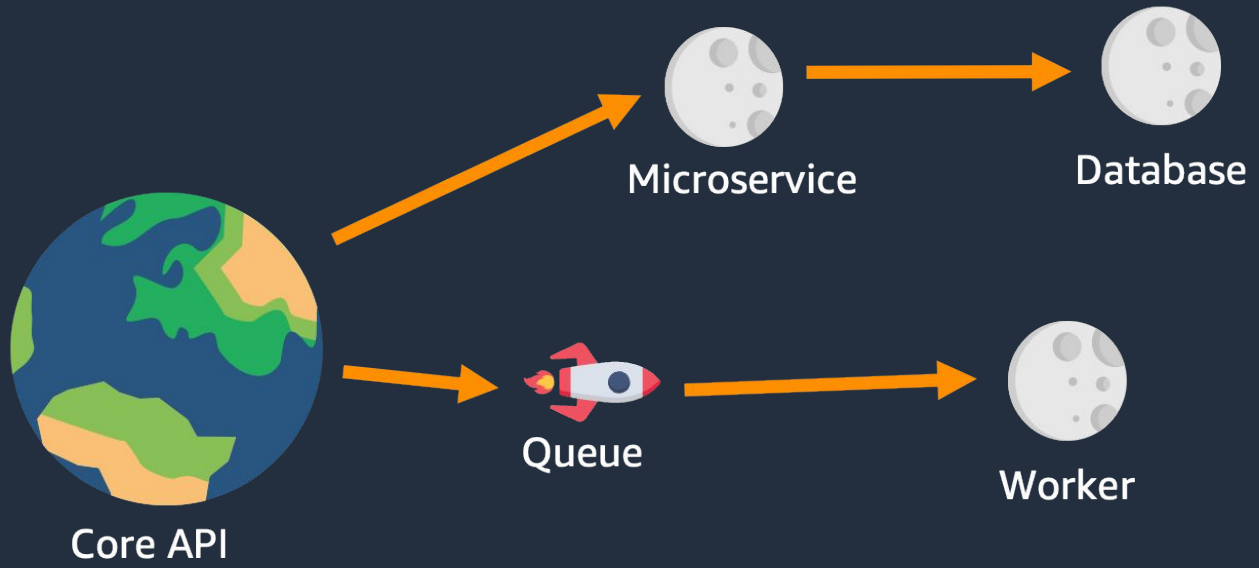
AWS Fargate

Serverless

- No servers to manage
- Pay only for resources when used
- Eliminate capacity planning
- Supports both Amazon EKS and Amazon ECS

And many customers run a mix of all three!

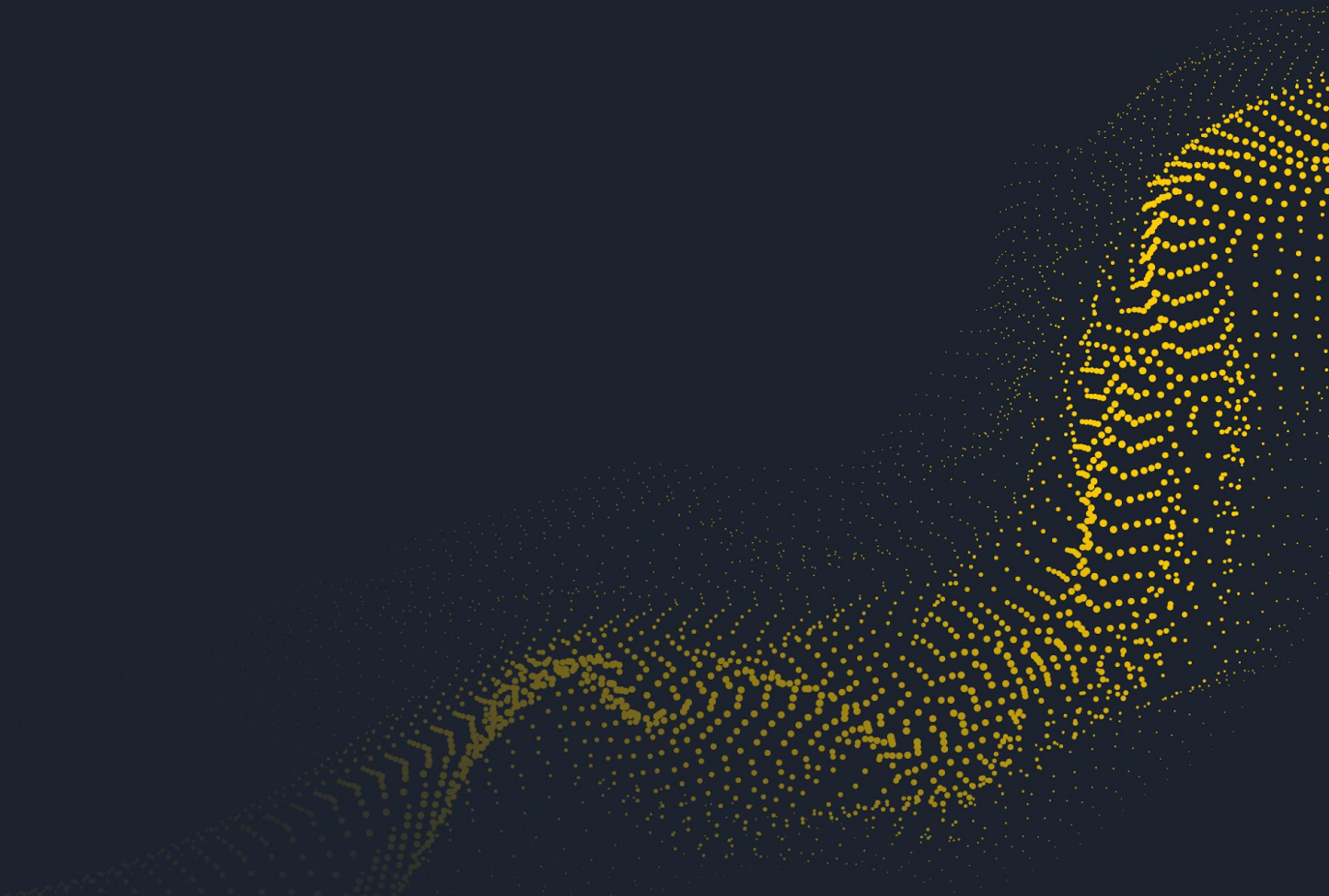
Decouple workloads responsibly.
And it is okay to have a central monolith!



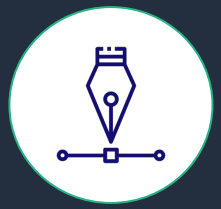
Amazon ECS & DevOps Terminology

- **Task definition**: The task definition is a text file, in JSON format that allows specifying details like launch type, CPU, etc that should be used with the containers in the task.
- **Task**: A task is the instantiation of a task definition within a cluster
- **Service**: An Amazon ECS service allows to run and maintain a specified number of instances of a task definition
- **Continuous Integration (CI)**: The practice of automating the integration of code changes from multiple contributors into a single software project
- **Continuous Delivery (CD)**: A methodology in which teams ensure that the application can be reliably released at any time, but which still relies on human intervention to determine what gets pushed into production.
- **Continuous Delivery Deployment (CDD)**: This process takes continuous delivery a step further by automatically deploying all releases into production

Demo



Launch a bare AWS ECS cluster



Configure AWS ECS repo & AWS CodeCommit

Deploy a Hello ECS task on the bare cluster

Configure AWS CodePipeline & Execution

Perform Rolling Update

- Github repo
- Blog

THANK YOU

aws