aws INNOVATE

MODERN APPLICATIONS EDITION

20 October, 2022

Best practices for deploying containers

Donnie Prakoso

Principal Developer Advocate, ASEAN
Amazon Web Services

https://donnie.id

@donnieprakoso

donnieprakoso

go.donnie.id/youtube

donnieprakoso

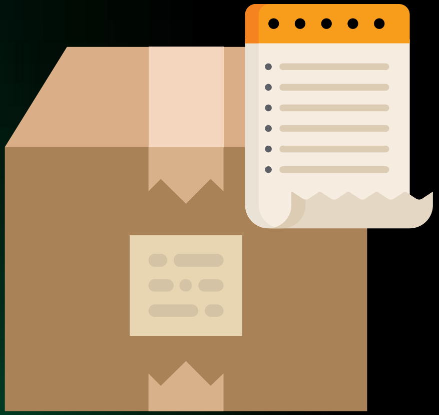
Key takeaways

- How to use AWS Copilot to deploy your application
- How to define and use config and secrets
- How to implement backing service and service discovery
- How to implement CI/CD with AWS Copilot
- How to implement Pub/Sub architecture
- How to run one-time task

Agenda

- Best Practices: Twelve-Factor App and Design Patterns
- Overview of Amazon ECS and AWS Fargate
- Build, release and operate containerized apps with AWS Copilot
- Demo, demo, and more demos!

Twelve-Factor App Principles



I. Codebase

II. Dependencies

III. Config

IV. Backing services

V. Build, release, run

VI. Processes

VII. Port binding

VIII. Concurrency

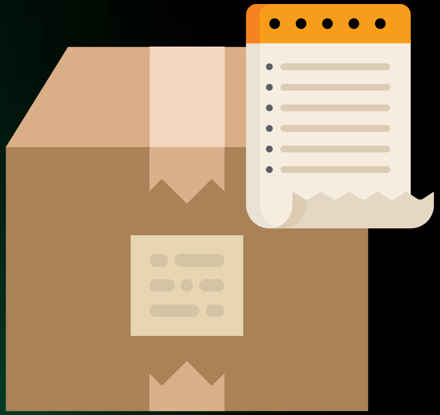
IX. Disposability

X. Dev/prod parity

XI. Logs

XII. Admin processes

Twelve-Factor App Principles



I. Codebase

II. Dependencies

III. Config

IV. Backing services

V. Build, release, run

VI. Processes

VII. Port binding

VIII. Concurrency

IX. Disposability

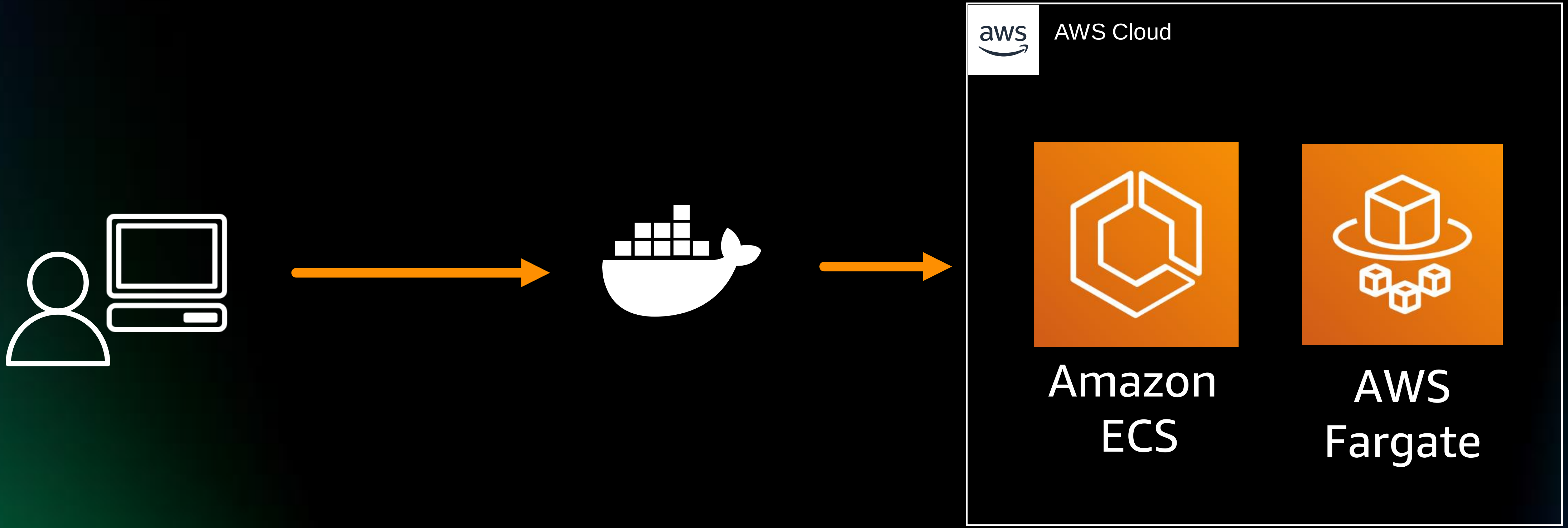
X. Dev/prod parity

XI. Logs

XII. Admin processes

Developing with containers

Typical Process to Ship App



Amazon Elastic Container Service (Amazon ECS)

- Container-level networking
- Advanced task placement
- Deep integration with AWS platform
- Amazon ECS CLI
- Global footprint
- Powerful scheduling engines
- Automatic scaling
- Amazon CloudWatch metrics
- Load balancers

AWS Fargate

AWS Fargate

Containers on demand

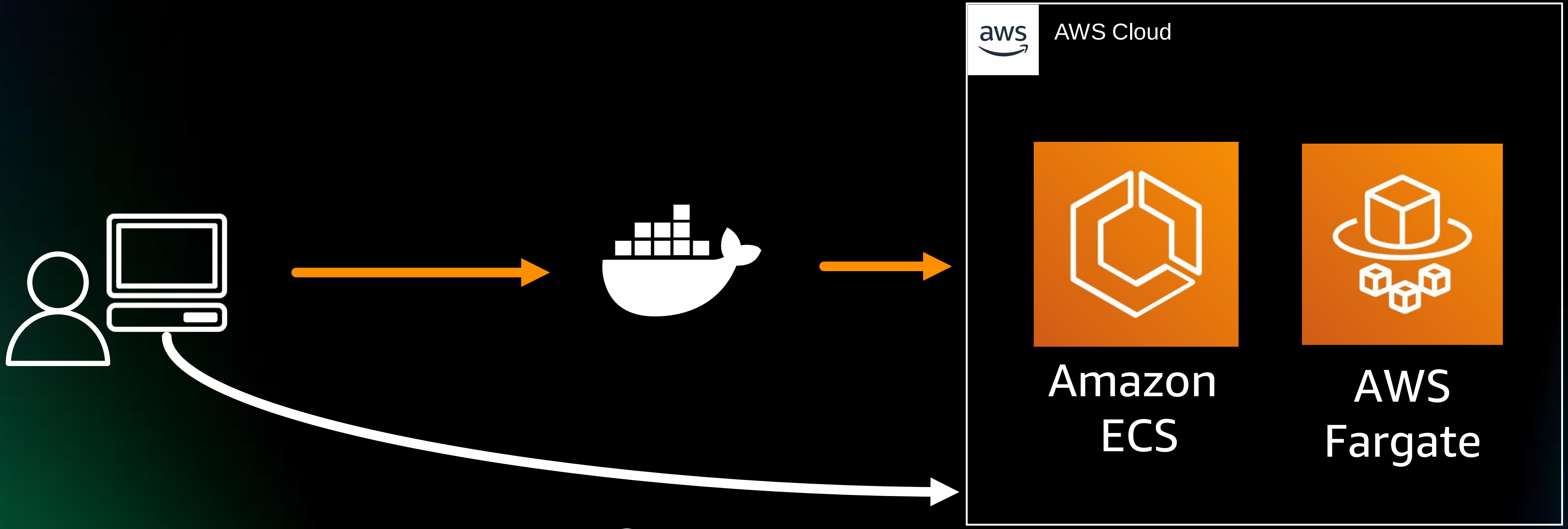
No infrastructure

Manage everything at container level

Launch quickly & scale easily

Resource-based pricing

Typical Process to Deploy App



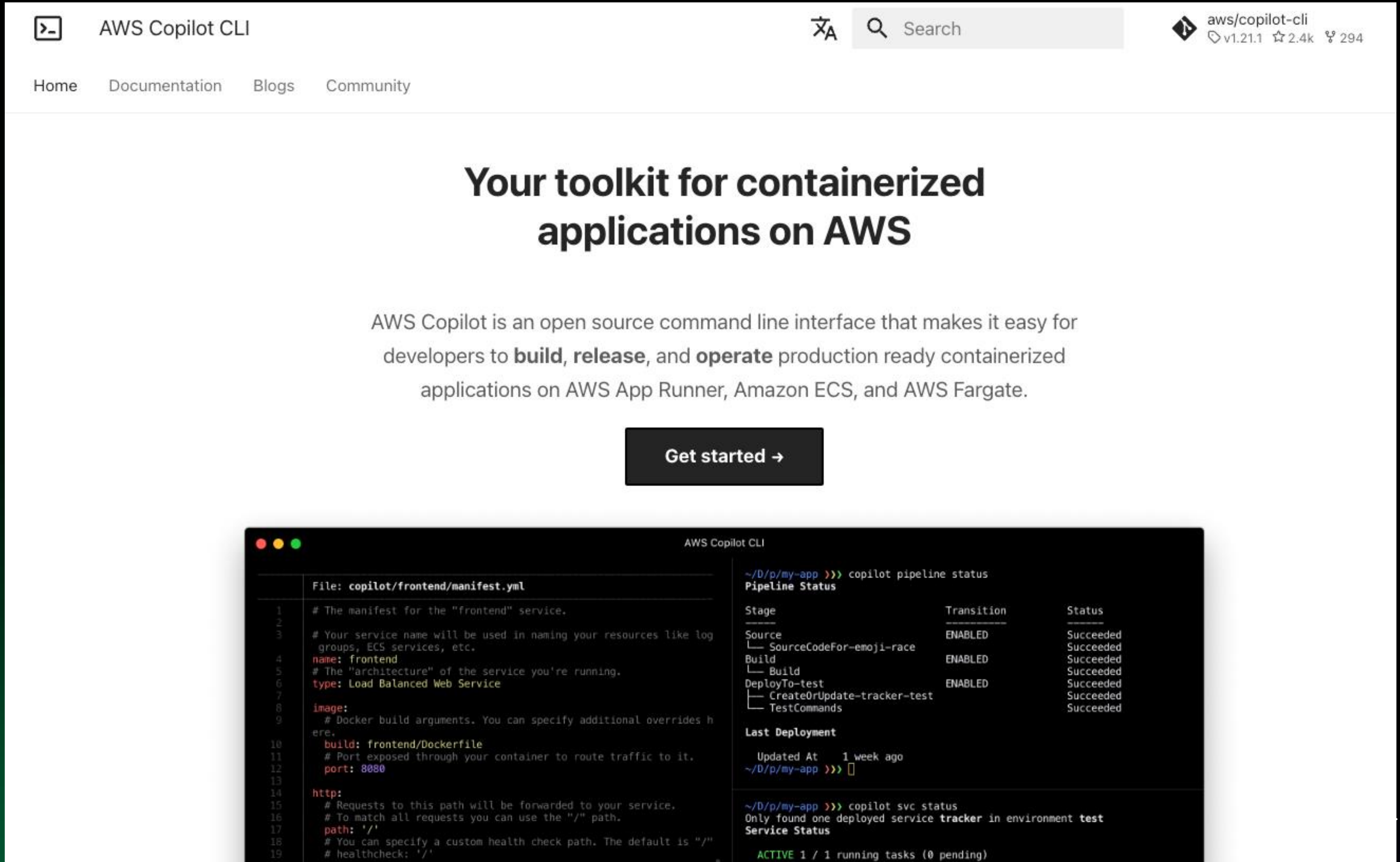
How?

Container Deployment Challenges

- How do I deploy applications?
- How do I add a service & integrate with AWS services?
- How can I test without affecting productions?
- How do I release applications?
- How to implement service discovery?

Introducing AWS Copilot CLI

https://aws.github.io/copilot-cli/



Your toolkit for containerized applications on AWS

AWS Copilot is an open source command line interface that makes it easy for developers to **build**, **release**, and **operate** production ready containerized applications on AWS App Runner, Amazon ECS, and AWS Fargate.

```
File: copilot/frontend/manifest.yml

# The manifest for the "frontend" service.

# Your service name will be used in naming your resources like log
 groups, ECS services, etc.
name: frontend
# The "architecture" of the service you're running.
type: Load Balanced Web Service

image:
  # Docker build arguments. You can specify additional overrides h
ere.
  build: frontend/Dockerfile
  # Port exposed through your container to route traffic to it.
  port: 8080

http:
  # Requests to this path will be forwarded to your service.
  # To match all requests you can use the "/" path.
  path: '/'
  # You can specify a custom health check path. The default is "/"
  # healthcheck: '/'
```

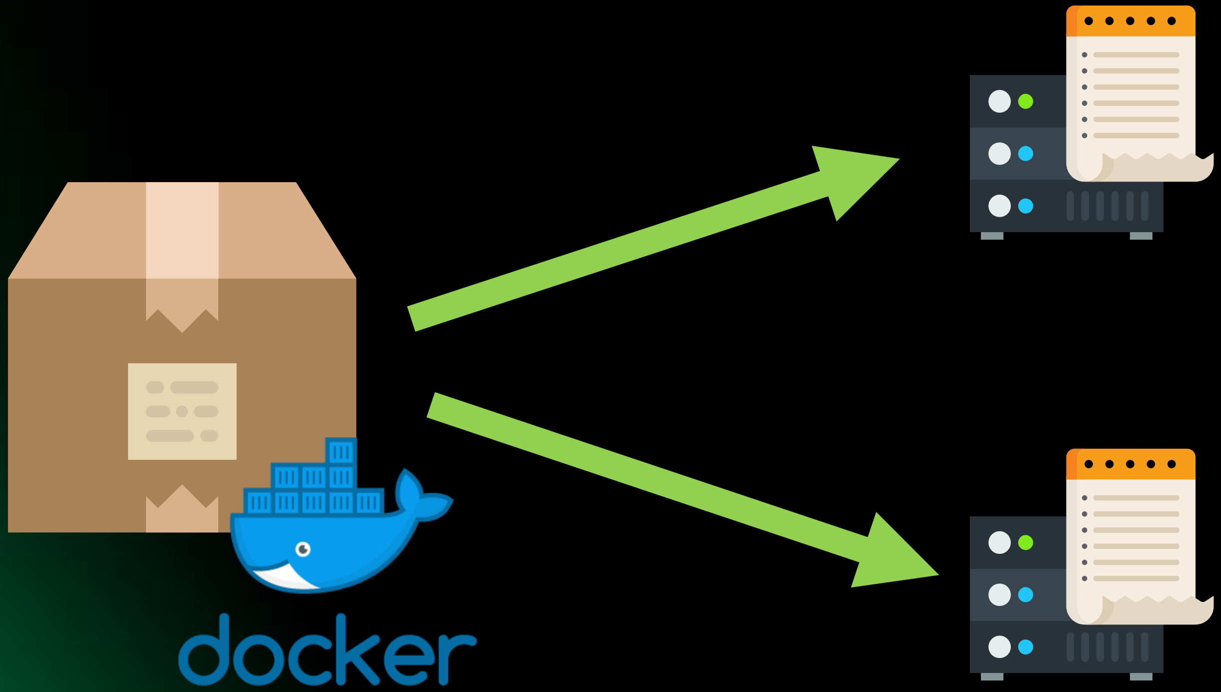
```
~/D/p/my-app ❯❯❯ copilot pipeline status
Pipeline Status

Stage                              Transition          Status
-----                              ----------          ------
Source                             ENABLED             Succeeded
└── SourceCodeFor-emoji-race                           Succeeded
Build                              ENABLED             Succeeded
└── Build                                              Succeeded
DeployTo-test                      ENABLED             Succeeded
├── CreateOrUpdate-tracker-test                        Succeeded
└── TestCommands                                       Succeeded

Last Deployment

  Updated At    1 week ago
~/D/p/my-app ❯❯❯ ▯
```

```
~/D/p/my-app ❯❯❯ copilot svc status
Only found one deployed service tracker in environment test
Service Status

  ACTIVE 1 / 1 running tasks (0 pending)
```

Demo 1

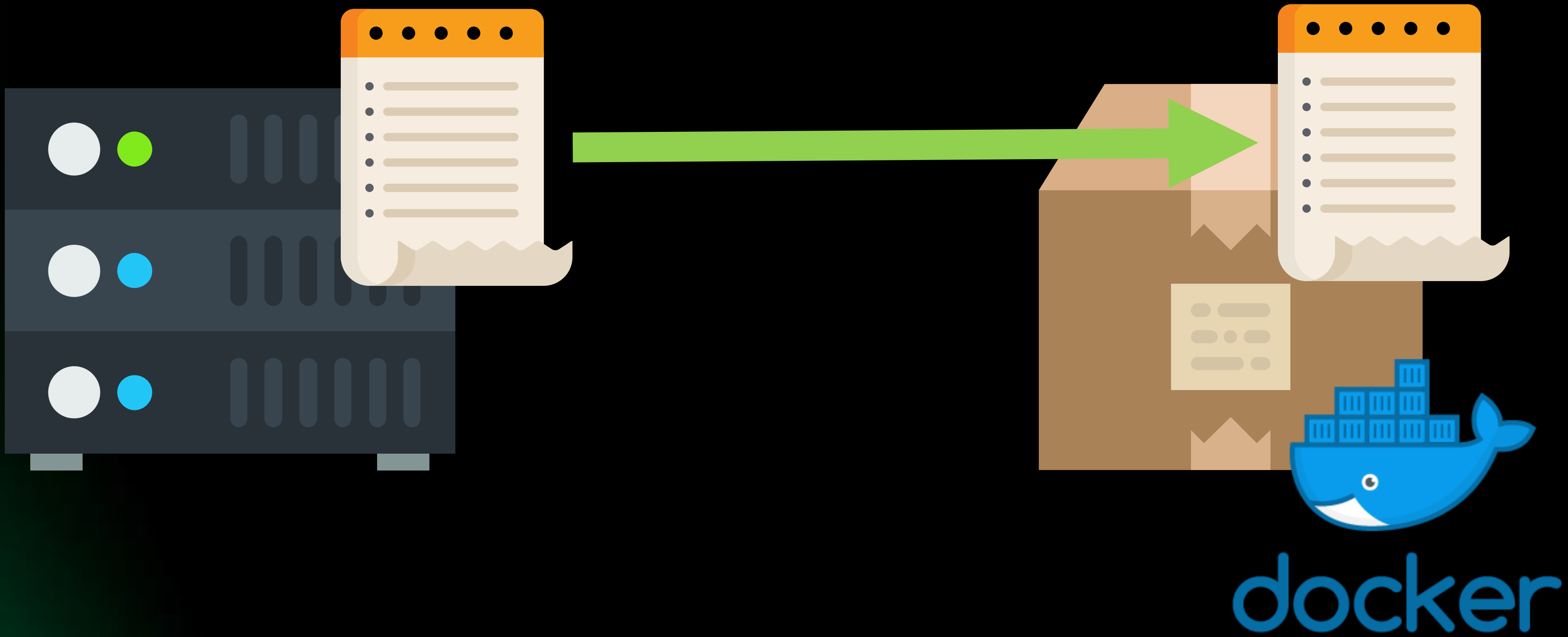
Twelve-Factor Application: Config

Same container deployed to both environments. Configuration is part of the environment on the host



- Development
- Production

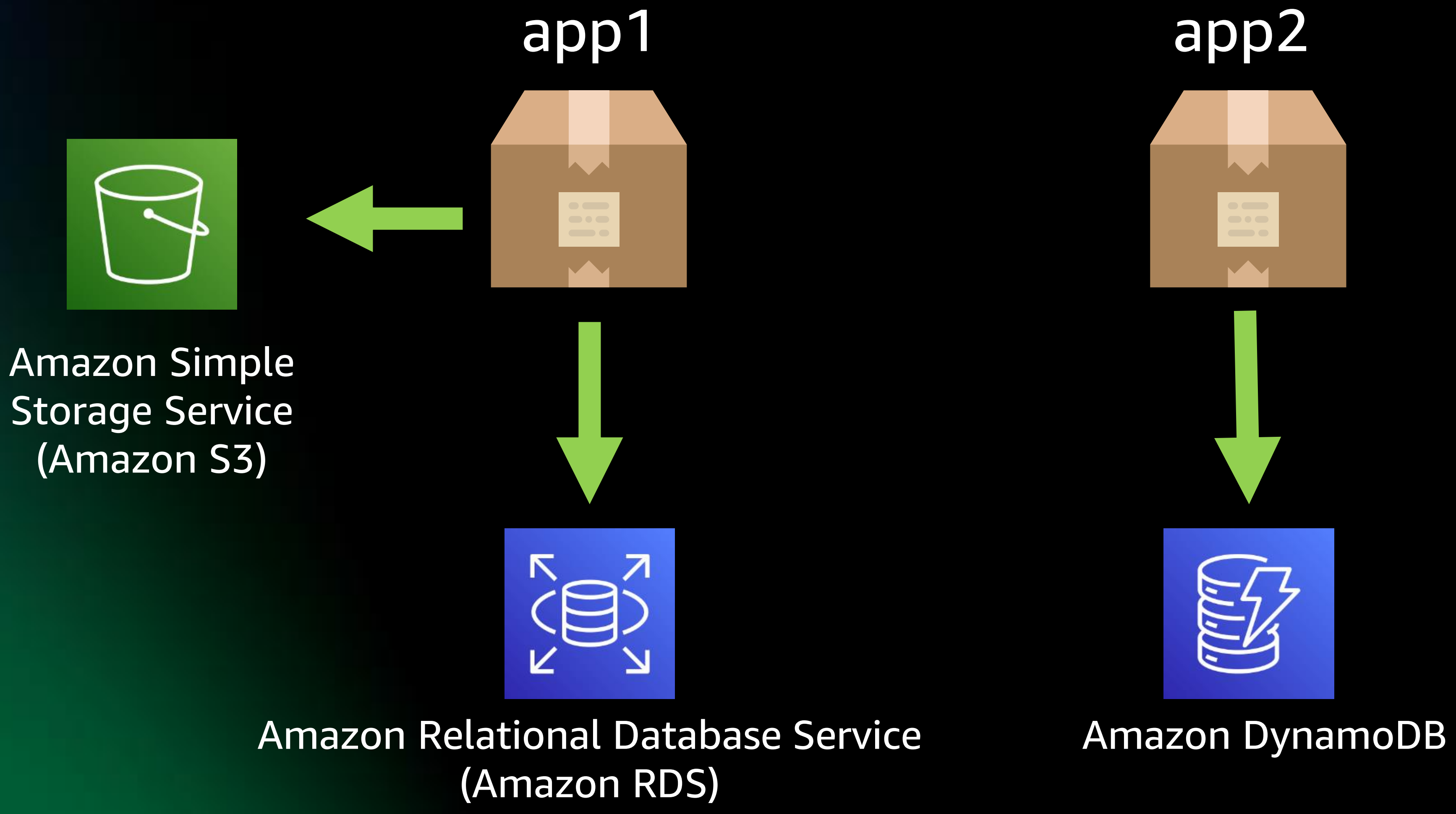
At runtime, the container gets config from the environment



Demo 2

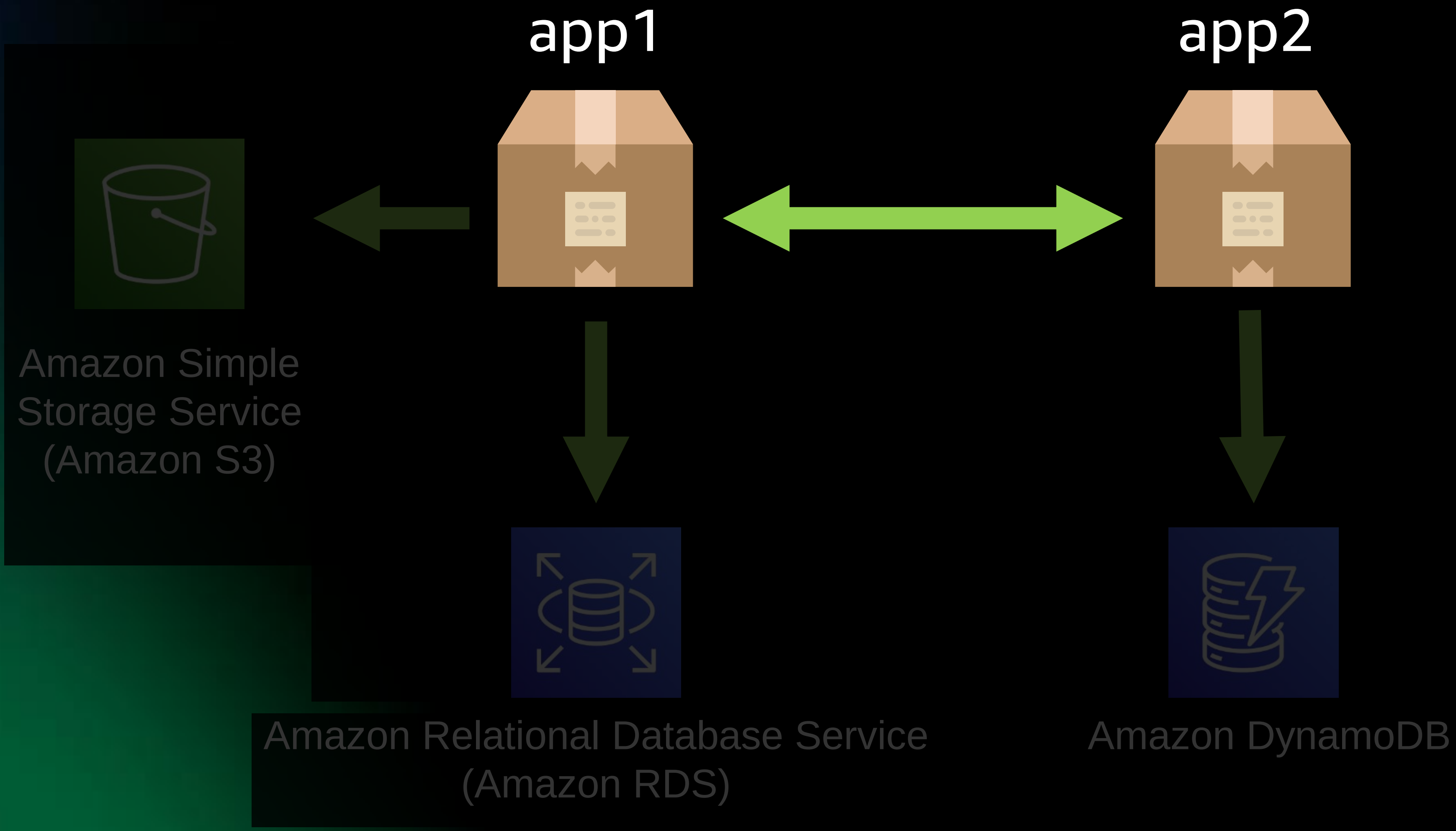
Twelve-Factor Application: Backing Services

Treat local services just like remote third party ones


app1
app2
Amazon Simple Storage Service (Amazon S3)
Amazon Relational Database Service (Amazon RDS)
Amazon DynamoDB

Service discovery

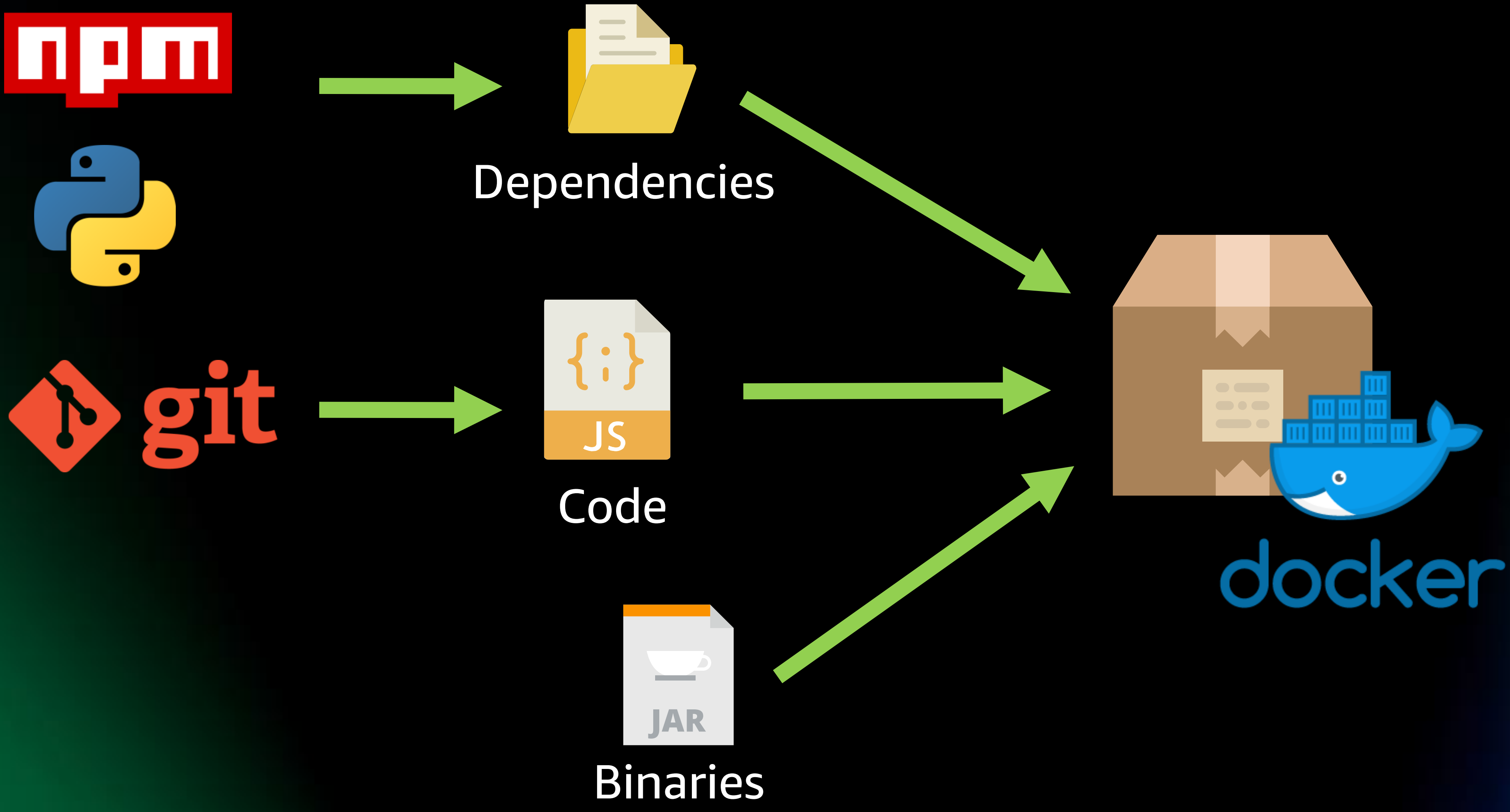
A mechanism for services to discover and interact with each other


app1
app2
Amazon Simple Storage Service (Amazon S3)
Amazon Relational Database Service (Amazon RDS)
Amazon DynamoDB

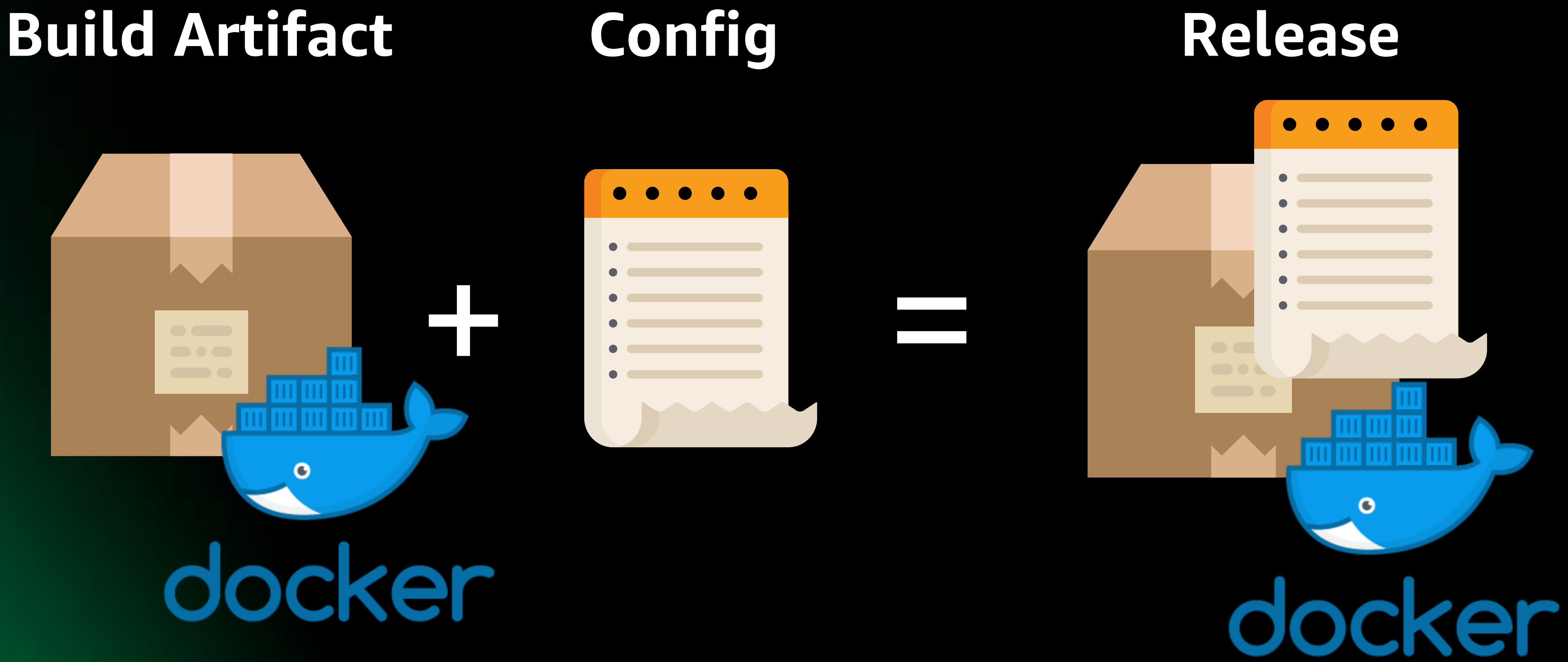
Demo 3

Twelve-Factor Application: Build, Release, Run

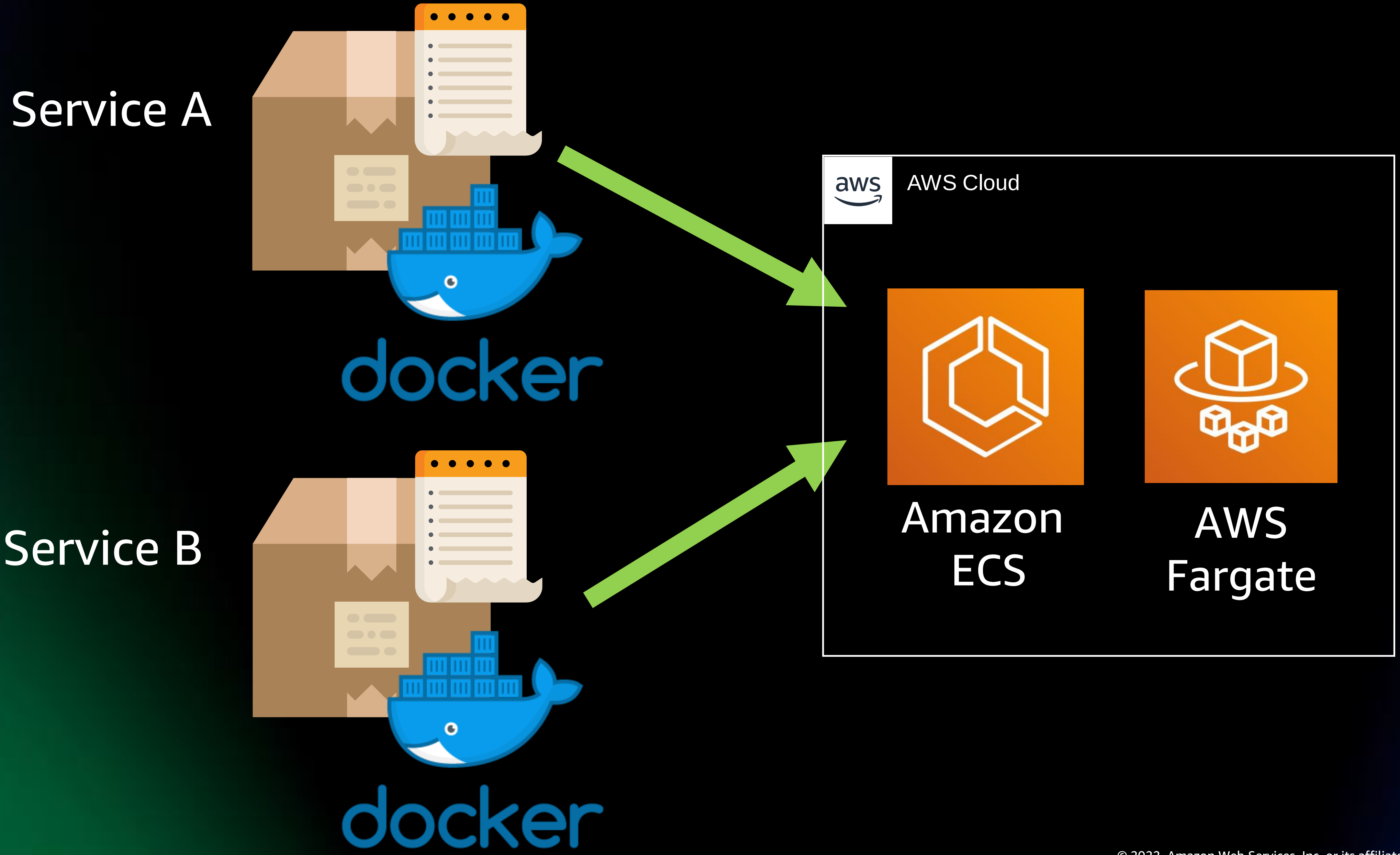
Build



Release



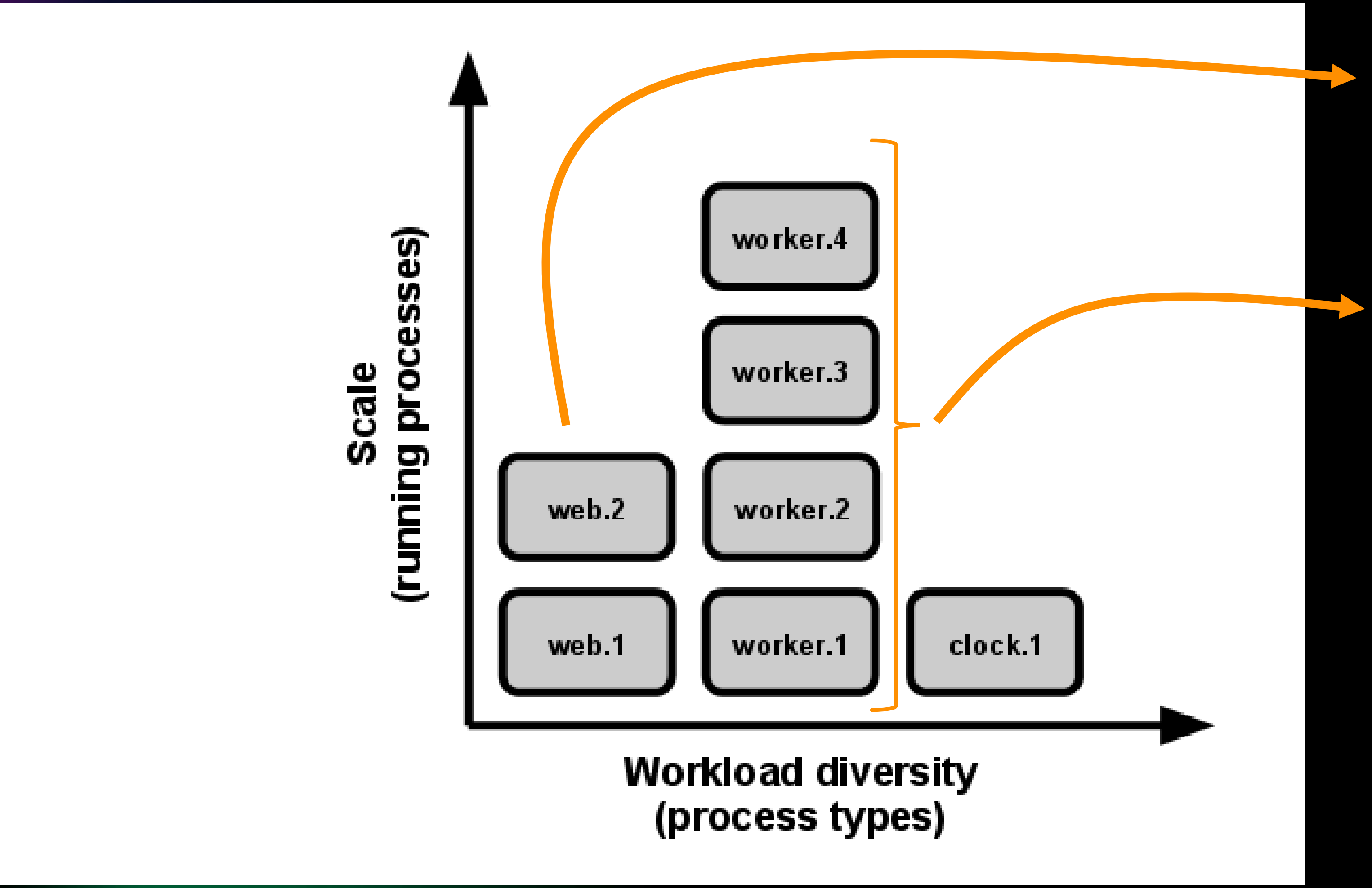
Run


Service A
docker
Service B
docker
aws
AWS Cloud
Amazon ECS
AWS Fargate

Demo 4

Twelve-Factor Application: Concurrency

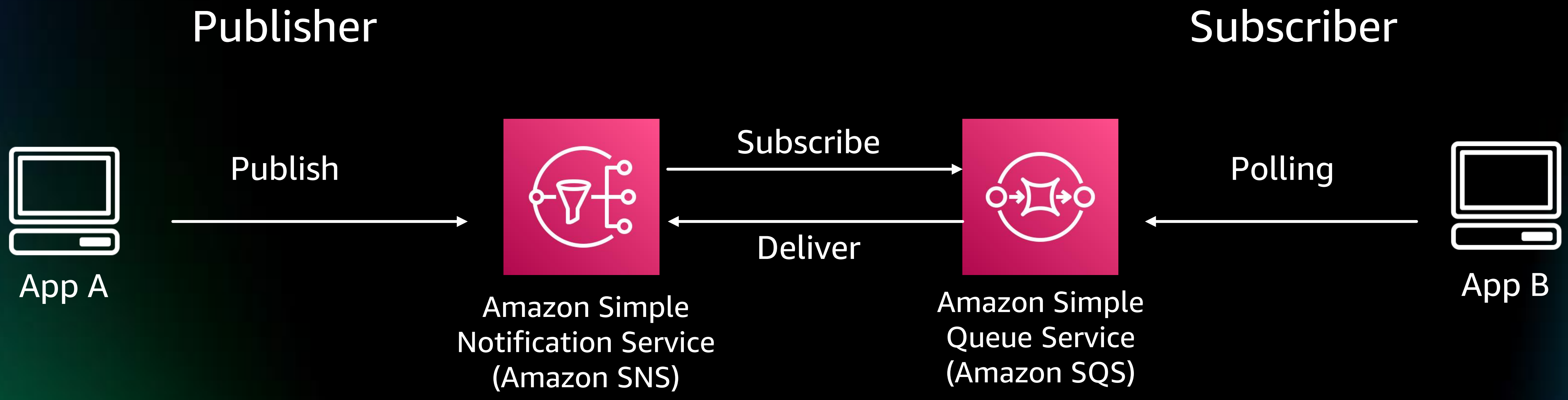
Concurrency



> Load Balanced Web Service

> Worker Service

Pub/Sub


Publisher
Subscriber
Publish
Subscribe
Deliver
Polling
App A
App B
Amazon Simple Notification Service (Amazon SNS)
Amazon Simple Queue Service (Amazon SQS)

Demo 5

Twelve-Factor Application: Admin Processes

Running Admin Processes

Admin / management processes are inevitable:

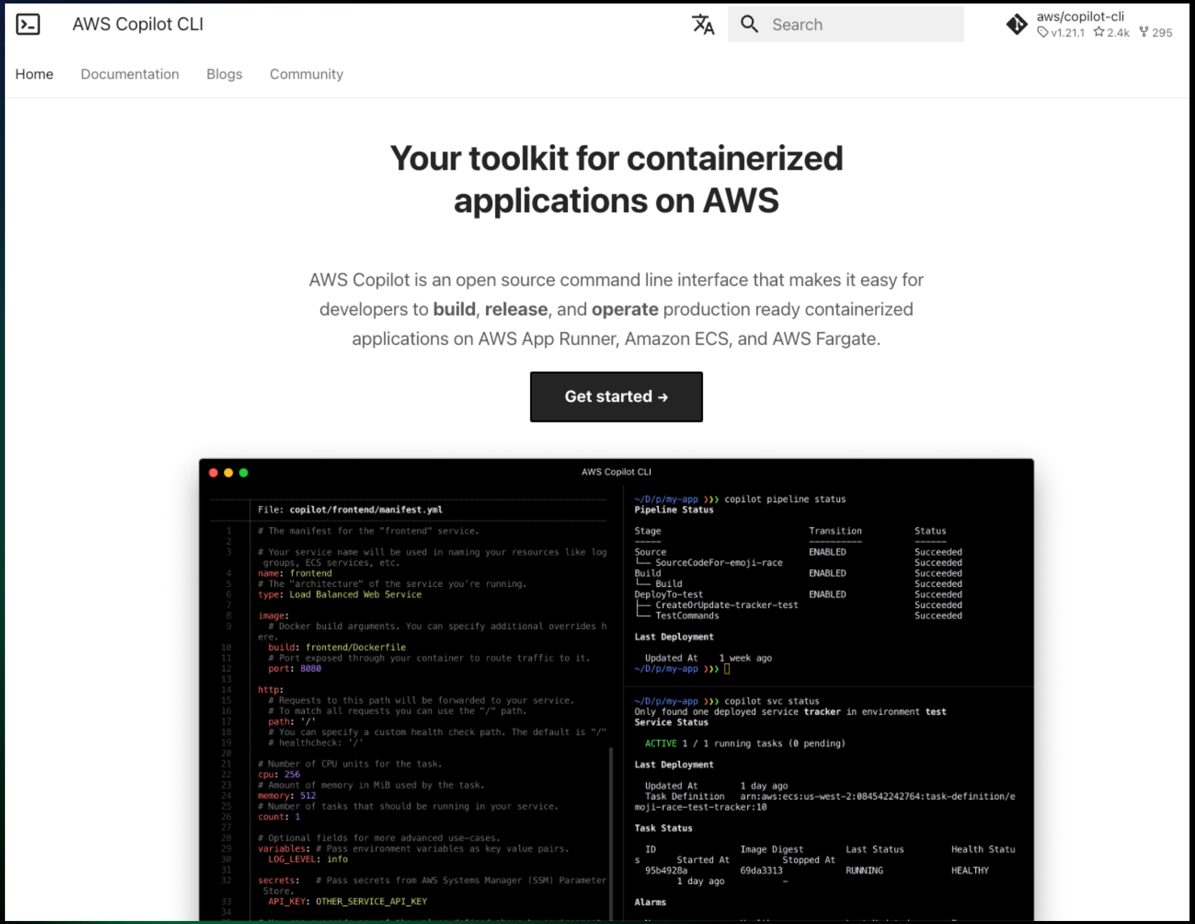
- Migrate database
- Repair some broken data
- Once a week move database records older than X to cold storage
- Every day email a report to this person

Demo 6

AWS Copilot resources

https://aws.github.io/copilot-cli/

https://copilot.rocks/



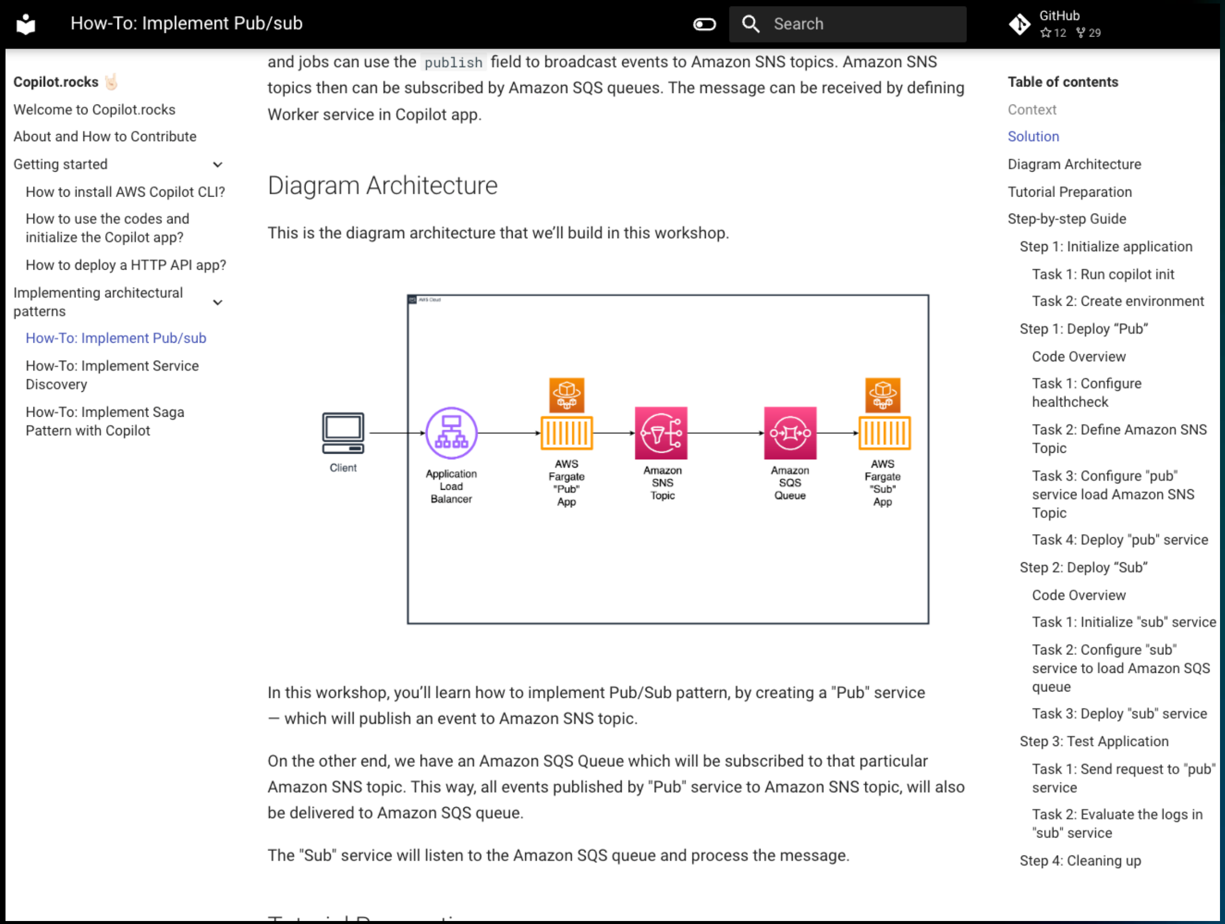
Your toolkit for containerized applications on AWS

AWS Copilot is an open source command line interface that makes it easy for developers to **build**, **release**, and **operate** production ready containerized applications on AWS App Runner, Amazon ECS, and AWS Fargate.

```yaml
File: copilot/frontend/manifest.yml

# The manifest for the "frontend" service.

# Your service name will be used in naming your resources like log
 groups, ECS services, etc.
name: frontend
# The "architecture" of the service you're running.
type: Load Balanced Web Service

image:
  # Docker build arguments. You can specify additional overrides h
ere.
  build: frontend/Dockerfile
  # Port exposed through your container to route traffic to it.
  port: 8080

http:
  # Requests to this path will be forwarded to your service.
  # To match all requests you can use the "/" path.
  path: '/'
  # You can specify a custom health check path. The default is "/"
  # healthcheck: '/'

# Number of CPU units for the task.
cpu: 256
# Amount of memory in MiB used by the task.
memory: 512
# Number of tasks that should be running in your service.
count: 1

# Optional fields for more advanced use-cases.
variables: # Pass environment variables as key value pairs.
  LOG_LEVEL: info

secrets:   # Pass secrets from AWS Systems Manager (SSM) Parameter
 Store.
  API_KEY: OTHER_SERVICE_API_KEY
```

```
~/D/p/my-app ❯❯❯ copilot pipeline status
Pipeline Status

Stage                              Transition          Status
-----                              ----------          ------
Source                             ENABLED             Succeeded
└── SourceCodeFor-emoji-race                           Succeeded
Build                              ENABLED             Succeeded
└── Build                                              Succeeded
DeployTo-test                      ENABLED             Succeeded
├── CreateOrUpdate-tracker-test                        Succeeded
└── TestCommands                                       Succeeded

Last Deployment

  Updated At    1 week ago
~/D/p/my-app ❯❯❯
```

```
~/D/p/my-app ❯❯❯ copilot svc status
Only found one deployed service tracker in environment test
Service Status

  ACTIVE 1 / 1 running tasks (0 pending)

Last Deployment

  Updated At        1 day ago
  Task Definition   arn:aws:ecs:us-west-2:084542242764:task-definition/e
moji-race-test-tracker:10

Task Status

  ID                Image Digest        Last Status         Health Statu
s       Started At          Stopped At
  95b4928a          69da3313            RUNNING             HEALTHY
        1 day ago           -

Alarms
```



Copilot.rocks 🤘
- Welcome to Copilot.rocks
- About and How to Contribute
- Getting started
  - How to install AWS Copilot CLI?
  - How to use the codes and initialize the Copilot app?
  - How to deploy a HTTP API app?
- Implementing architectural patterns
  - How-To: Implement Pub/sub
  - How-To: Implement Service Discovery
  - How-To: Implement Saga Pattern with Copilot

and jobs can use the `publish` field to broadcast events to Amazon SNS topics. Amazon SNS topics then can be subscribed by Amazon SQS queues. The message can be received by defining Worker service in Copilot app.

Diagram Architecture

This is the diagram architecture that we'll build in this workshop.

In this workshop, you'll learn how to implement Pub/Sub pattern, by creating a "Pub" service — which will publish an event to Amazon SNS topic.

On the other end, we have an Amazon SQS Queue which will be subscribed to that particular Amazon SNS topic. This way, all events published by "Pub" service to Amazon SNS topic, will also be delivered to Amazon SQS queue.

The "Sub" service will listen to the Amazon SQS queue and process the message.

Table of contents
- Context
- Solution
- Diagram Architecture
- Tutorial Preparation
- Step-by-step Guide
  - Step 1: Initialize application
    - Task 1: Run copilot init
    - Task 2: Create environment
  - Step 1: Deploy "Pub"
    - Code Overview
    - Task 1: Configure healthcheck
    - Task 2: Define Amazon SNS Topic
    - Task 3: Configure "pub" service load Amazon SNS Topic
    - Task 4: Deploy "pub" service
  - Step 2: Deploy "Sub"
    - Code Overview
    - Task 1: Initialize "sub" service
    - Task 2: Configure "sub" service to load Amazon SQS queue
    - Task 3: Deploy "sub" service
  - Step 3: Test Application
    - Task 1: Send request to "pub" service
    - Task 2: Evaluate the logs in "sub" service
  - Step 4: Cleaning up

Visit the Modern Applications resource hub

Dive deeper with these resources to help you develop an effective plan for your modernization journey.

- Build modern applications on AWS
- Business value of cloud modernization
- An introduction to event-driven architectures
- Accelerate full-stack web and mobile app development
- Determining the total cost of ownership: Comparing serverless and server-based technologies
- Building event-driven architectures with AWS
- Continuous learning, continuous modernization

Visit resource hub

https://tinyurl.com/modern-apps-aws

AWS Training and Certification

Get started with Free Digital Training for you and your team today

Achieve key milestones and plan your next steps with the AWS Modern Application skills training

Access 500+ free digital courses with AWS Skill Builder

Earn an industry-recognized credential:
AWS Certified Developer – Associate
AWS Certified DevOps – Professional

Create a self-paced learning roadmap
AWS ramp-up guide - Developer
AWS ramp-up guide - DevOps

Thank you for attending AWS Innovate Modern Applications Edition

We hope you found it interesting! A kind reminder to **complete the survey.**
Let us know what you thought of today's event and how we can improve the event experience for you in the future.

aws-apj-marketing@amazon.com

twitter.com/AWSCloud

facebook.com/AmazonWebServices

youtube.com/user/AmazonWebServices

slideshare.net/AmazonWebServices

twitch.tv/aws

Thank you!

Donnie Prakoso
Principal Developer Advocate, ASEAN

Amazon Web Services

https://donnie.id

@donnieprakoso

donnieprakoso

go.donnie.id/youtube

donnieprakoso