ENGINEERING KIOSK ALPS MEETUP INNSBRUCK – 22.02.2024

From containers to serverless functions

Maximilian Schellhorn

Senior Solutions Architect
Amazon Web Services (AWS)

About me

Software Engineer 10+ years (WS, Catalysts, Ada Health, Zalando)

Since 2021 AWS Solutions Architect

Specialized on SaaS, Serverless, Java and EDA

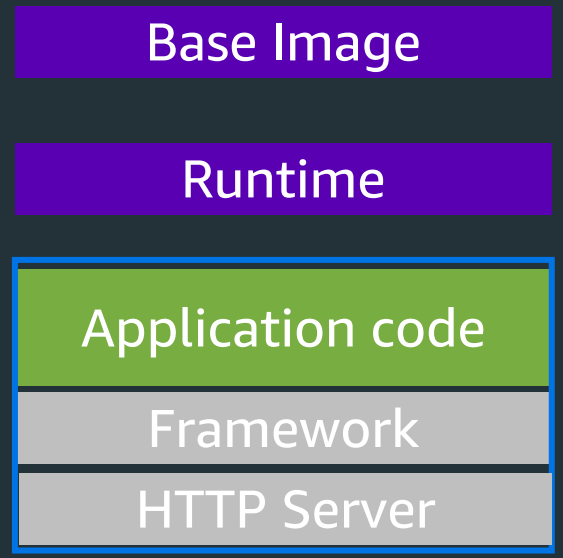
Public Speaking (Devoxx Belgium, JFokus, re:Invent)

Fundamentals, Performance & Scaling

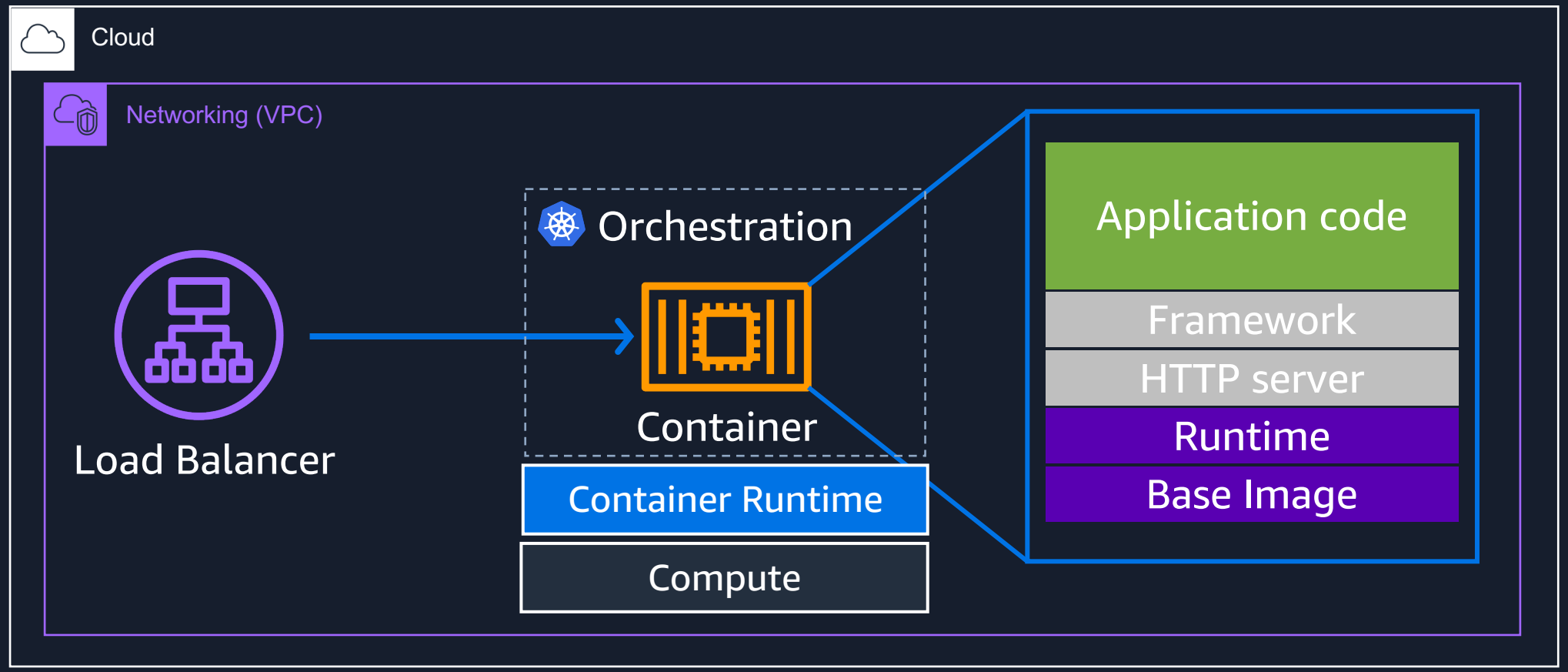
Containers or Serverless functions

Container image

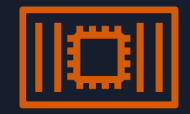
```
# Use Amazon Linux 2023 as base image
FROM amazonlinux:2023

# Install necessary dependencies and runtime
RUN yum install -y java-21-amazon-corretto

# Copy the local jar file into the container
COPY your-app.jar your-app.jar

# Expose port 8080
EXPOSE 8080

# Command to run the jar file
CMD ["java", "-jar", "your-app.jar"]
```



Typical containerized application

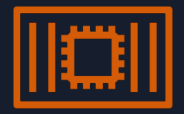


Container start up and scaling

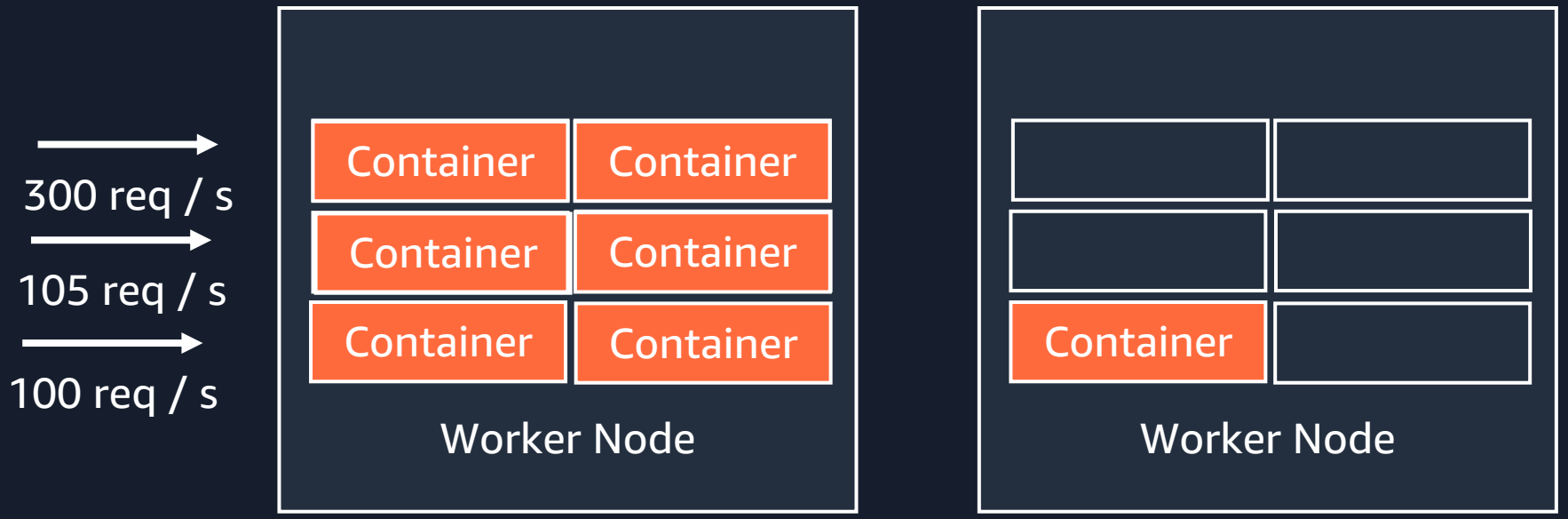


- Initialize the container once
- Handle multiple request with the same instance (concurrently)
- Container (usually) keeps running after request processing

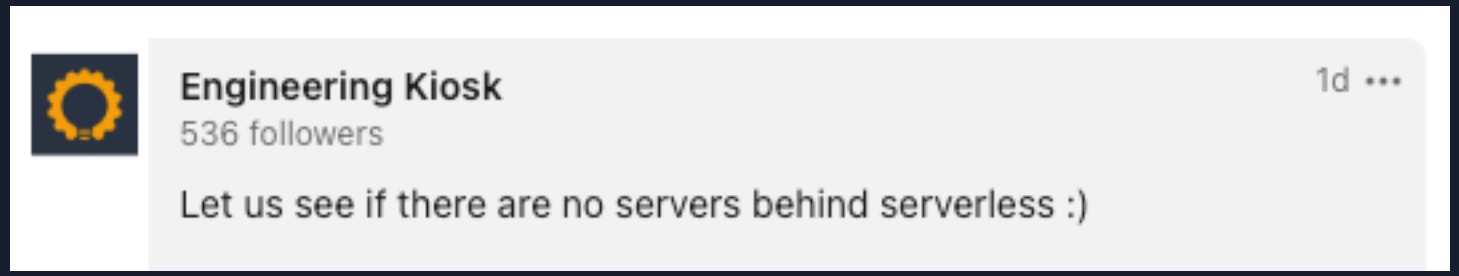
Container scaling



- Container based scaling (Example: Kubernetes Pods)
- Scaling based on metrics or manually (coarse grained)
- Node based scaling (if applicable)
- Optimized Auto-Scaling tools available (Karpenter, Cluster Auto Scaler)



Containers or Serverless functions



Containers or Serverless functions

Anatomy of a function

Handler function

- Function written in Java, JS, Python etc.
- Input params with req. information

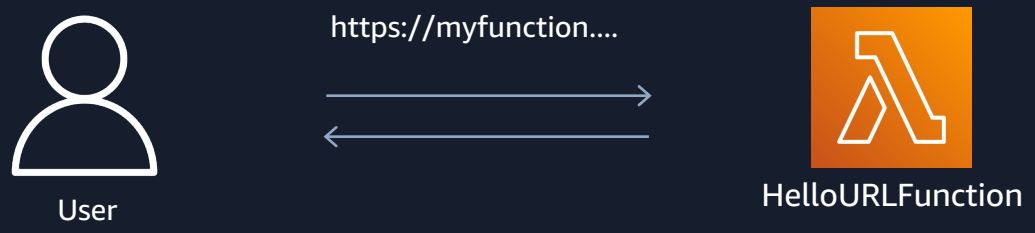
```
public ResponseEvent handleRequest(RequestEvent input, Context context)
  // Do stuff here with the input and context

  return ResponseEvent.Builder()
            .withBody("Return Something")
            .withStatusCode(200)
            .build();
}
```

Configuration and Deployment

- Runtime Version: Java 11, Java 17, Java 21, Node.js (16, 18, 20)
- Memory setting, CPU Architecture (x86, ARM), Timeout up to 15 minutes
- Package as zip archive or jar and upload UI, CLI or Terraform

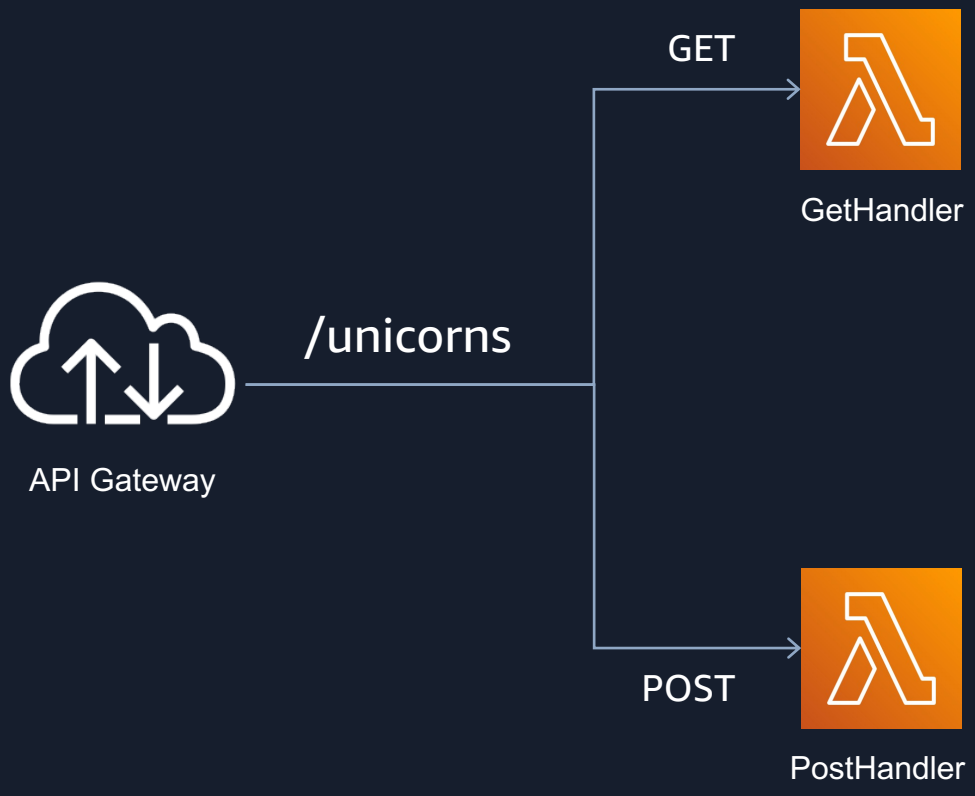
Serverless Functions

Example: Simple Function URL




```java
public class UnicornRequestIdHandler {
    private final Logger logger = LoggerFactory.getLogger(UnicornRequestIdHandler.class);

    public String handleRequest(Map<String, Object> input, Context context) {
        var requestId = context.getAwsRequestId();
        logger.info("Received input with %s".formatted(input));

        return "Received request with id: %s".formatted(requestId);
    }
}
```

Example: REST API



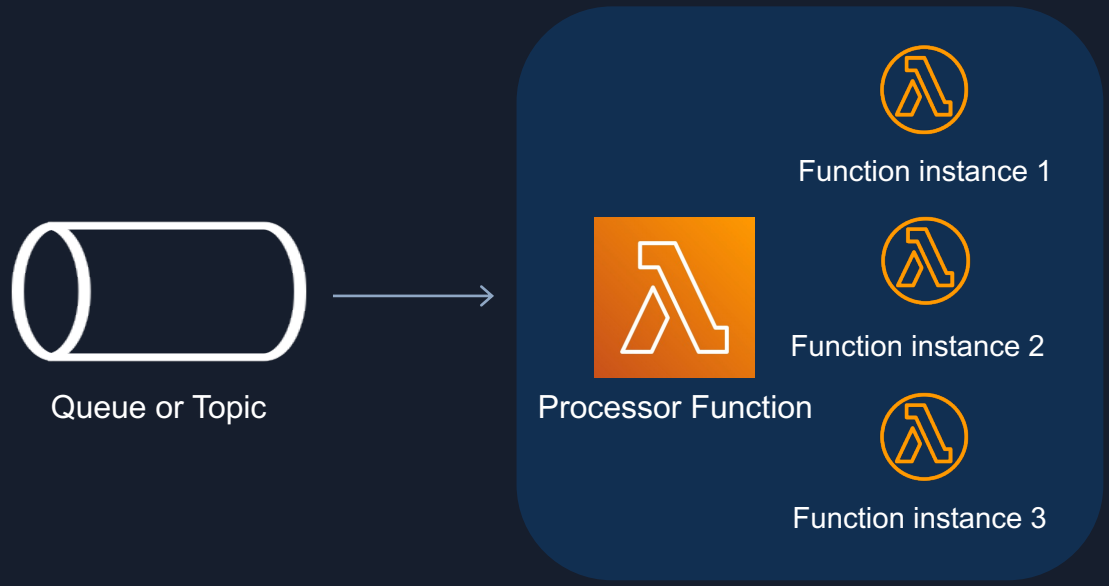
com.unicorn.store.GetHandler

```java
@Override
public APIGatewayV2HTTPResponse handleRequest(APIGatewayV2HTTPEvent event, /*Context */) {
    var unicorn = unicornService.get(event.getPathParameters().get("id"));

    return APIGatewayV2HTTPResponse.builder()
            .withStatusCode(200)
            .withBody(getJsonString(unicorn))
            .build();
}
```

```java
@Override
public APIGatewayV2HTTPResponse handleRequest(APIGatewayV2HTTPEvent event /*Context*/)  {
    var savedUnicorn = unicornService.save(getUnicornFromEvent(event));

    return APIGatewayV2HTTPResponse.builder()
            .withStatusCode(201)
            .withBody(getJsonString(savedUnicorn))
            .build();
}
```

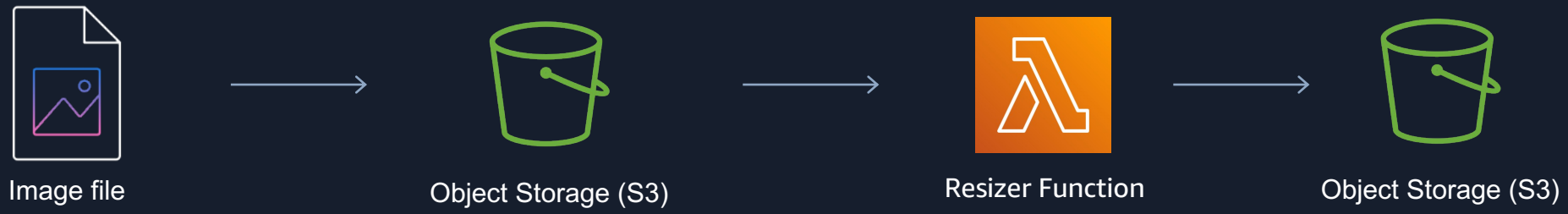
com.unicorn.store.PostHandler

Example: Queue processor



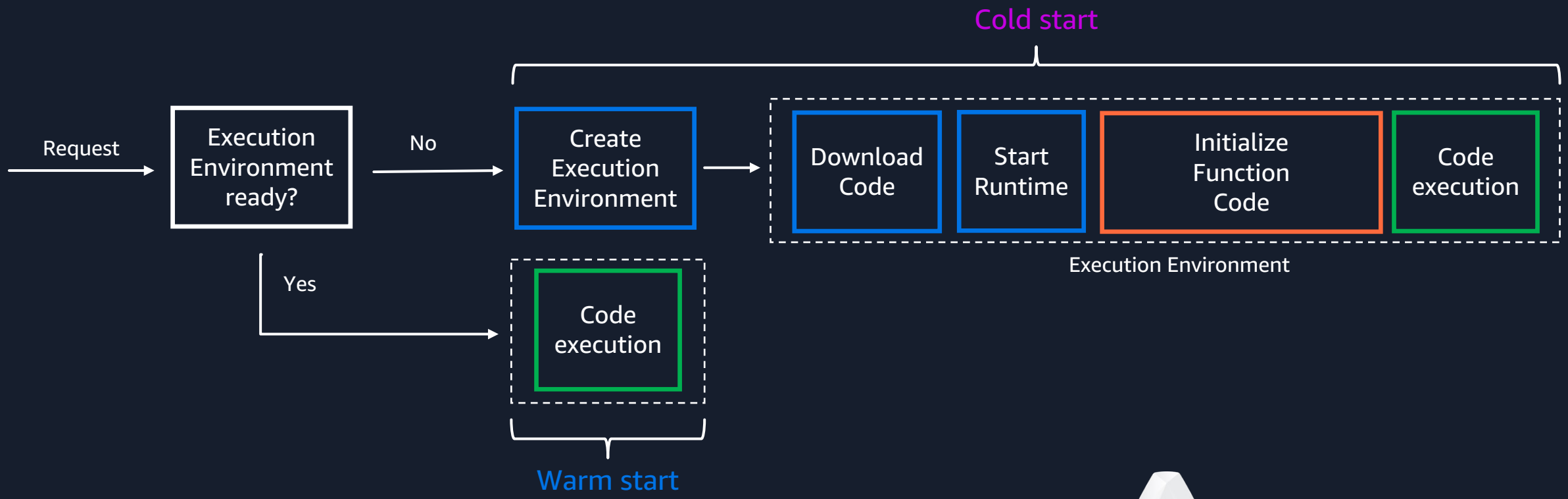

```java
@Override
public SQSBatchResponse handleRequest(SQSEvent sqsEvent, Context context) {
    var messages = sqsEvent.getRecords()
                    .stream()
                    .map(SQSMessage::getBody)
                    .toList();

    return processMessages(messages);
}
```

Example: Resizing images




```
public class UnicornDocumentHandler implements RequestHandler<S3Event, String> {

    @Override
    public String handleRequest(S3Event input, Context context) {}
}
```

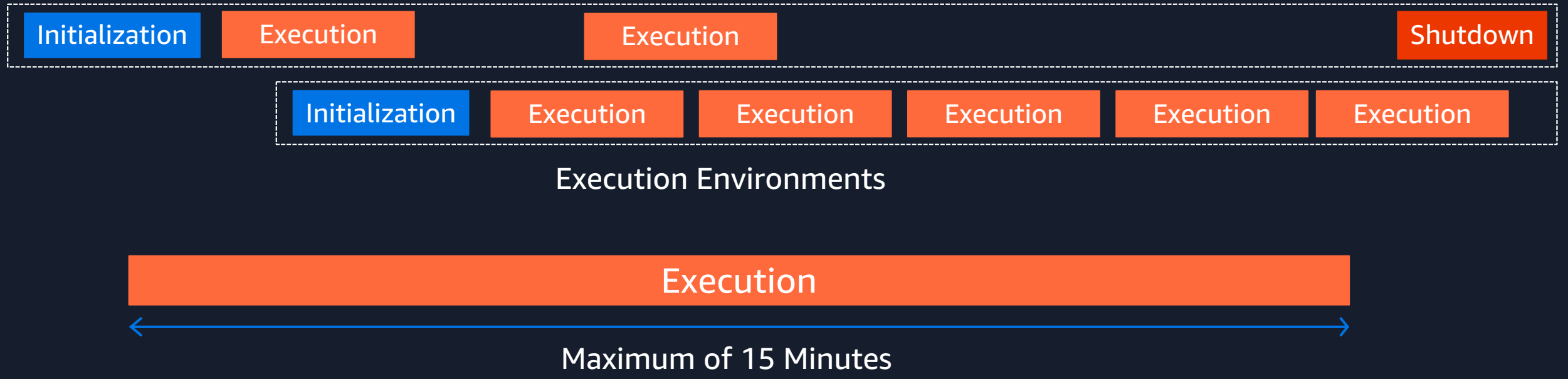
Example: Serverless function startup (Lambda)



Firecracker
microVM technology

Serverless Function environment

- „Short“ lived and ephemeral
- Execution environments can be shut down after inactivity
- In Memory state will be lost – use external cache or persistent storages



Maintenance, Security & Operations

Serverless Function management

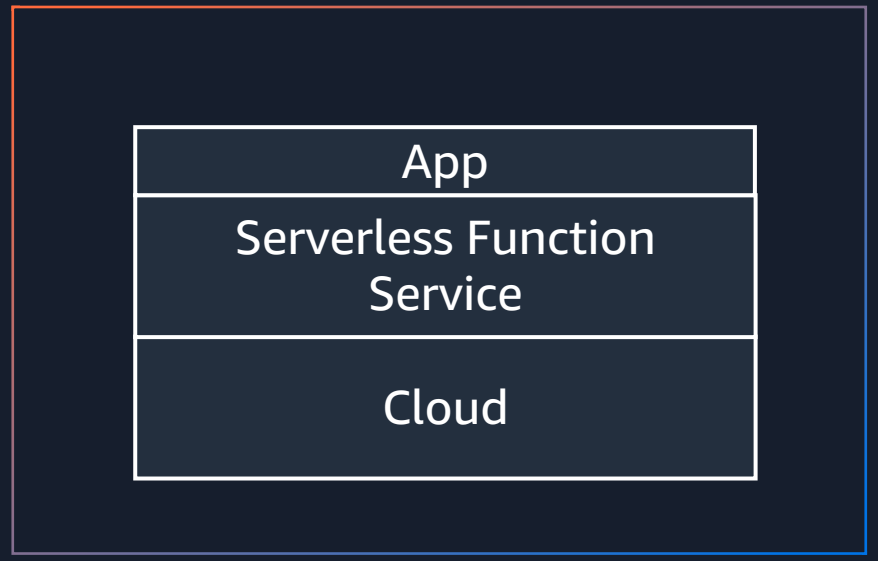
High availability and scaling by design

Automatic patching of runtime (Log4Shell)

Logs, Metrics, Traces out of the box

No access to underlying hosts

Limited debugging capabilities out of the box



Container management

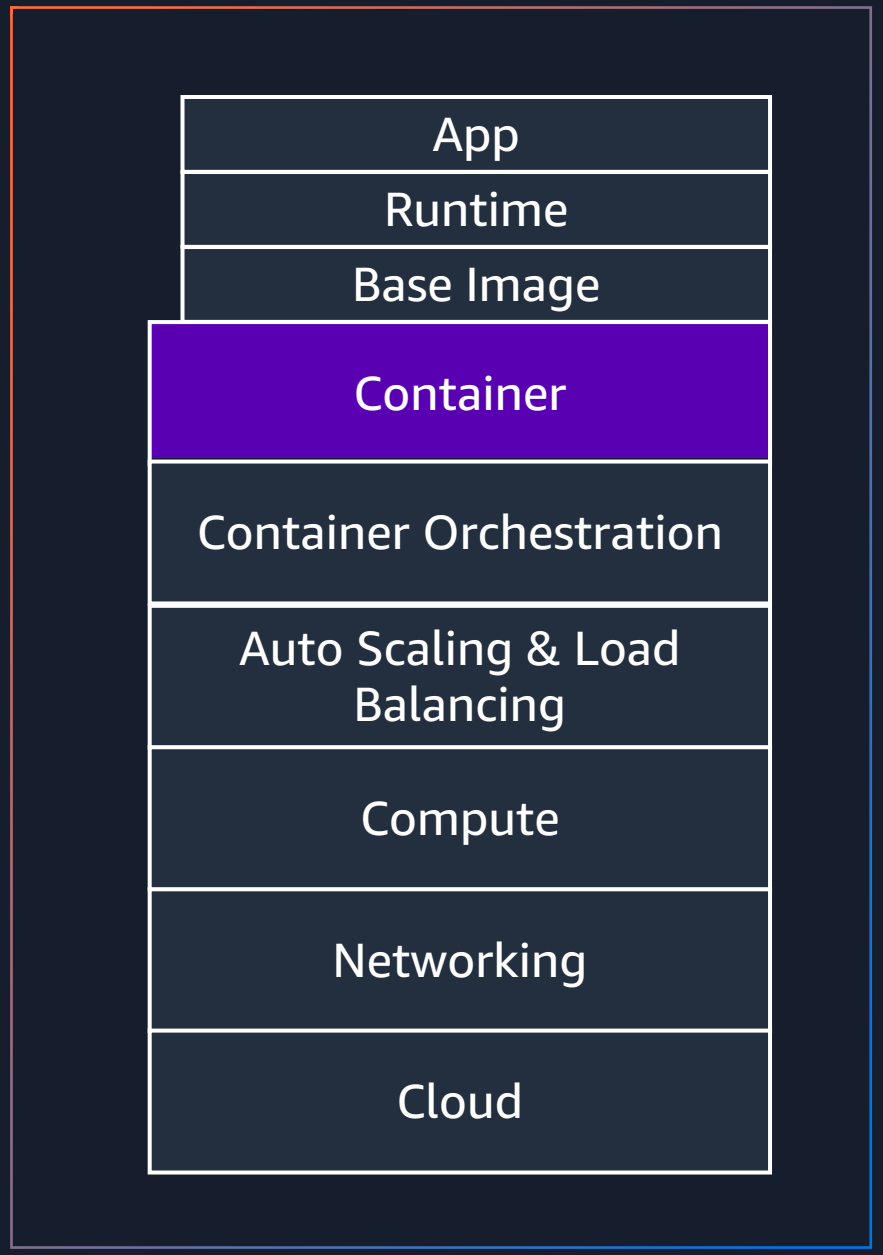
Multiple levels of abstractions possible

More control and debugging capabilities

Responsibility for patching OS, Base Images (CVEs)

Cluster upgrades (Kubernetes)

Platform teams provide abstractions for devs



What should devs do apart from writing code?

Cost efficiency

Serverless functions cost efficiency

Pay per use - Advantages for spikey and idle workloads

Scale to 0 = Zero cost

Efficiently experiment globally

Low management overhead – time to market

Function

Frankfurt

Function

Singapore

Function

US

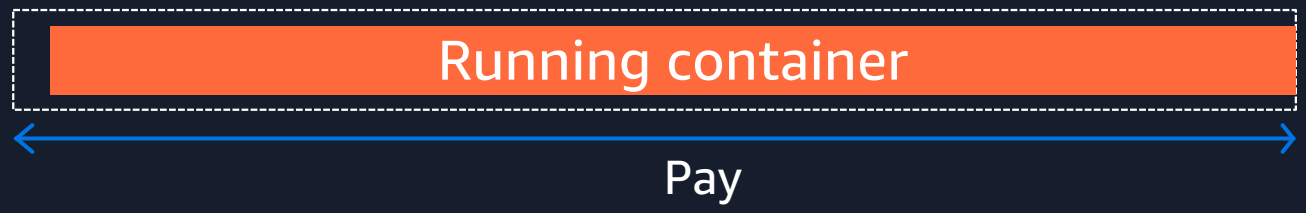
Container cost efficiency

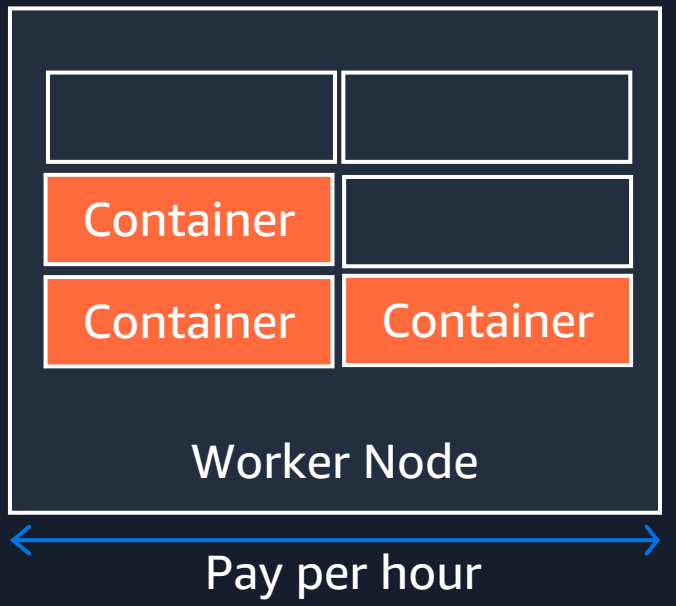
Predictable pricing

Cost efficient for constant load (Own car vs. Rental car)

Optimize via instance type, size, CPU architecture (ARM64 / x86)

Additional cost for management of the infrastructure





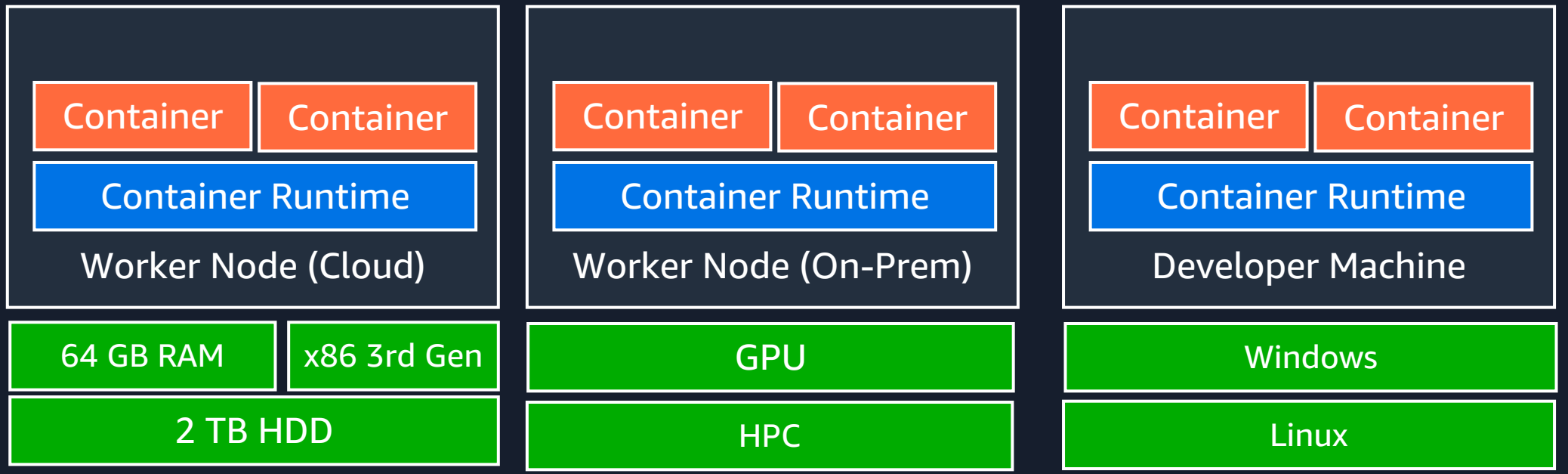
Pay for instance vs. pay per use

Flexibility vs. Simplicity

Containers offer flexbility

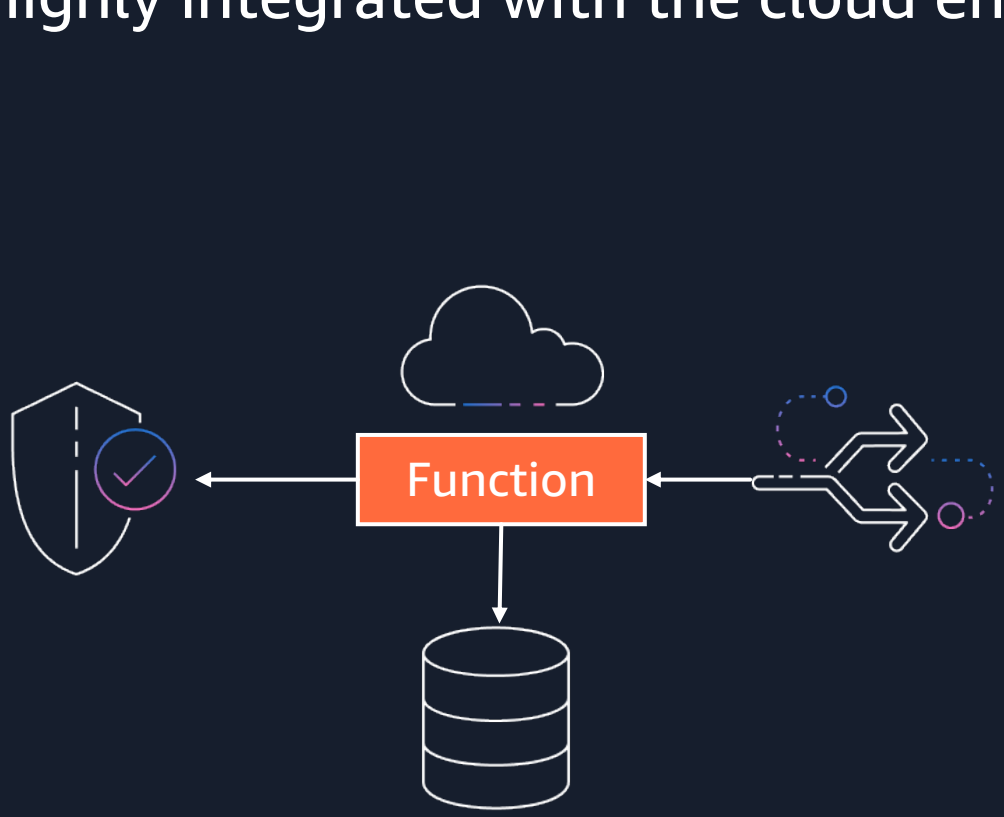
Broad choice from CNCF projects & large open source community

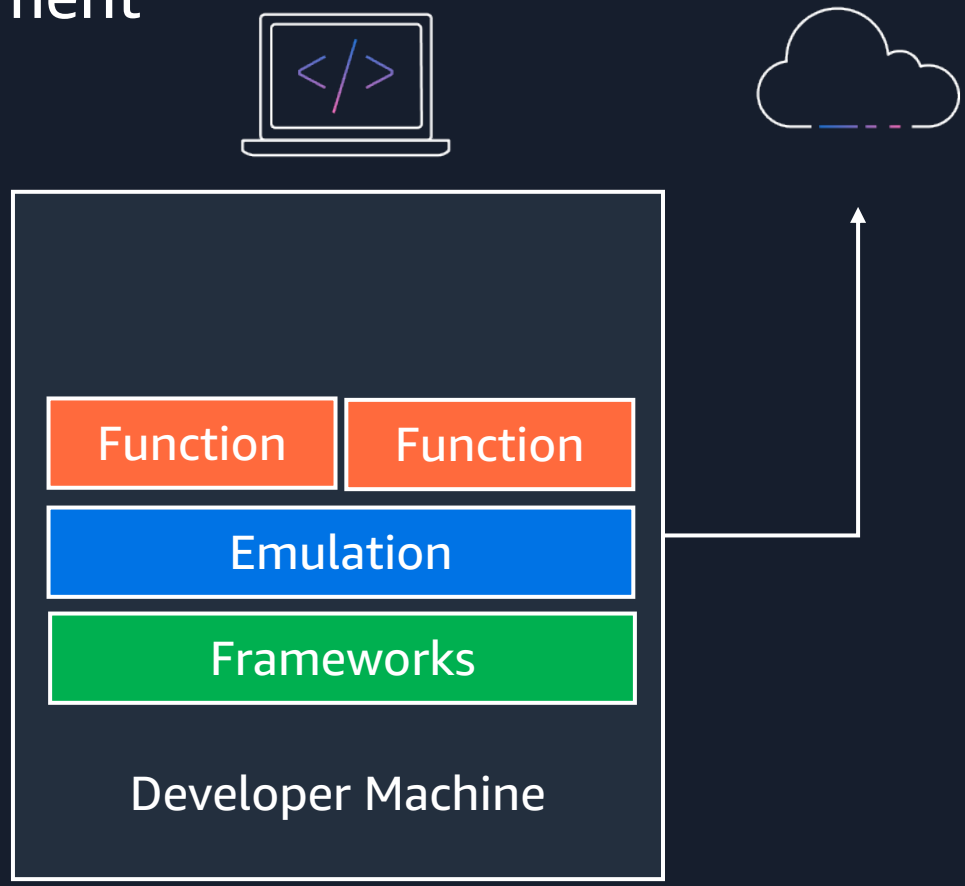
Flexibility in the choice of hardware



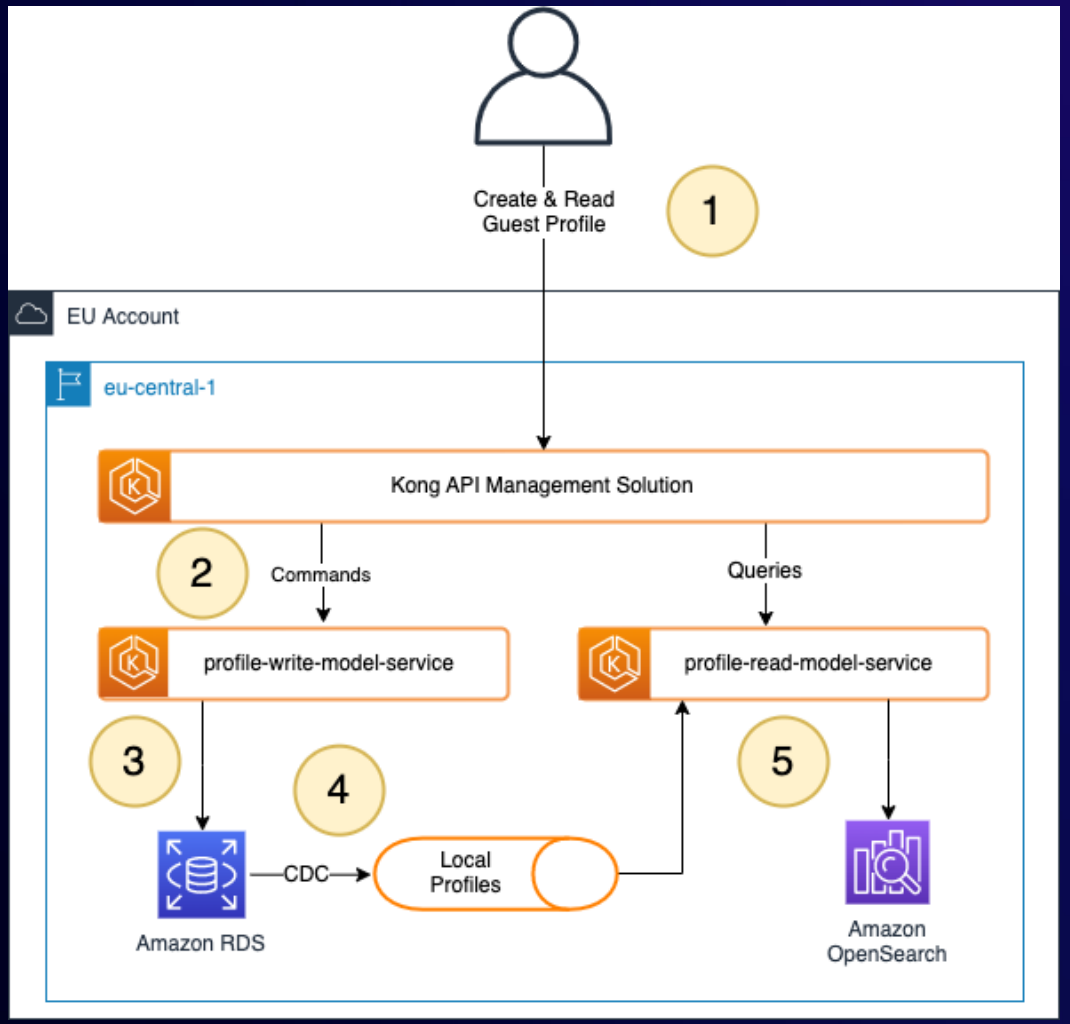
Functions offer simplicity

Basic choice of Hardware (Memory, CPU architecture and temp storage)
Deploy to cloud or local emulation – no on-premise
Highly integrated with the cloud environment



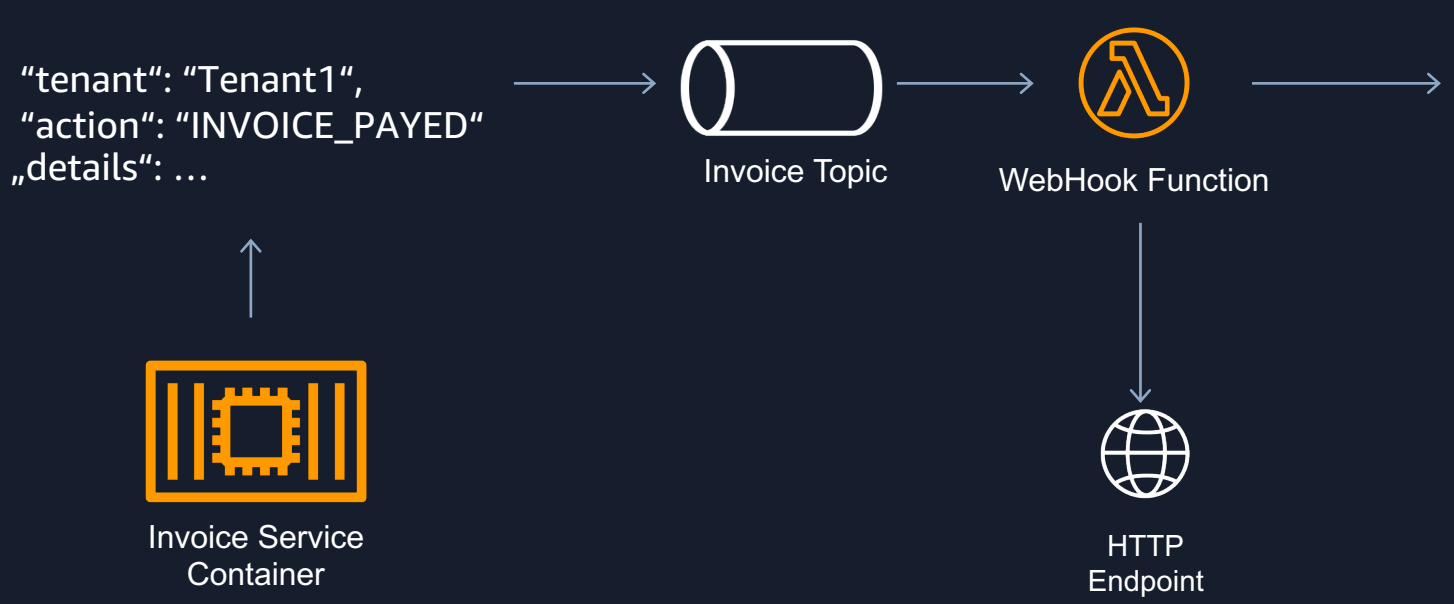


Example 1: Global SaaS provider (Containers)



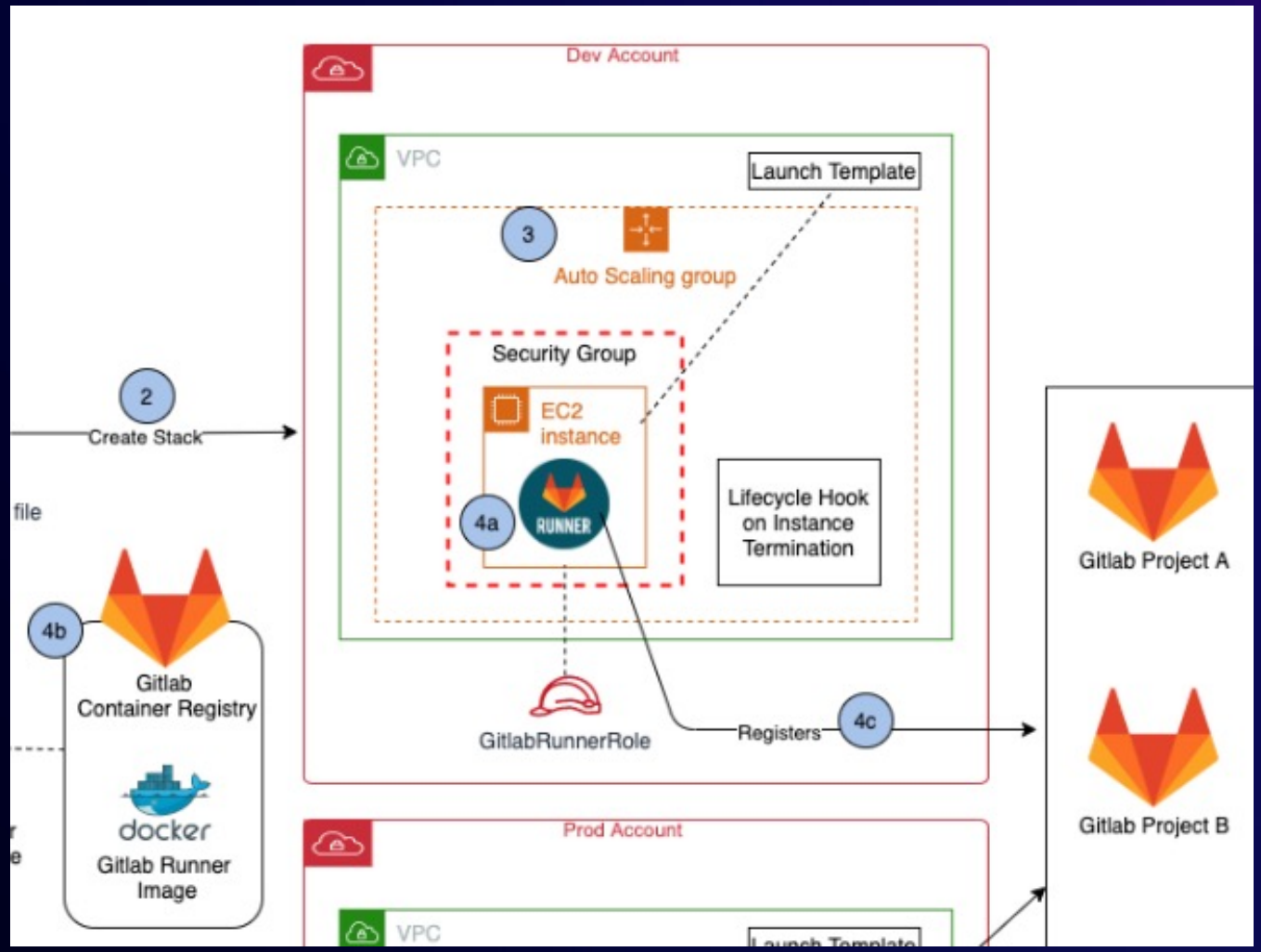
https://aws.amazon.com/blogs/architecture/how-shiji-group-created-a-global-guest-profile-store-on-aws/

Example 2: Serverless Webhooks (Both)



TENANT (PK)	ACTION (SK)	URL
TENANT#1	INVOICE_PAYED	crm.tenant-1.com
TENANT#2	INVOICE_UPLOADED	erp.tenant-2.com

Example 3: CI/CD pipeline (Containers)



https://aws.amazon.com/blogs/devops/deploy-and-manage-gitlab-runners-on-amazon-ec2/

Thank you!

Maximilian Schellhorn

LinkedIn: linkedin.com/in/maxschell/

Twitter: @maschnetwork

What we couldn’t cover today

- Serverless Containers
- Not entirely serverless concepts such as OpenFaaS and Knative
- Granularity of Serverless Functions (How big should a function be?)
- Advanced options such as Snapshotting and Provisioned Concurrency
- What about lock-in / Switching cost (Hexagonal Architecture)
- Other Serverless components (Databases, Queues etc.)

Upcoming events in Tyrol

Uni Innsbruck – Tech Lab (15.03.2024)
Building Event-Driven Architectures in the cloud
Presentation + Hands-On Labs

Kitzbühel (25 – 26 April 2024)
Moderne Anwendungen in der Cloud
Presentation