aws SUMMIT

Reactive Microservices Architecture on AWS

Sascha Möllering

Solutions Architect, @sascha242, Amazon Web Services Germany GmbH

Why are we here today?

https://secure.flickr.com/photos/mgifford/4525333972

Agenda

What is Reactive Architecture?

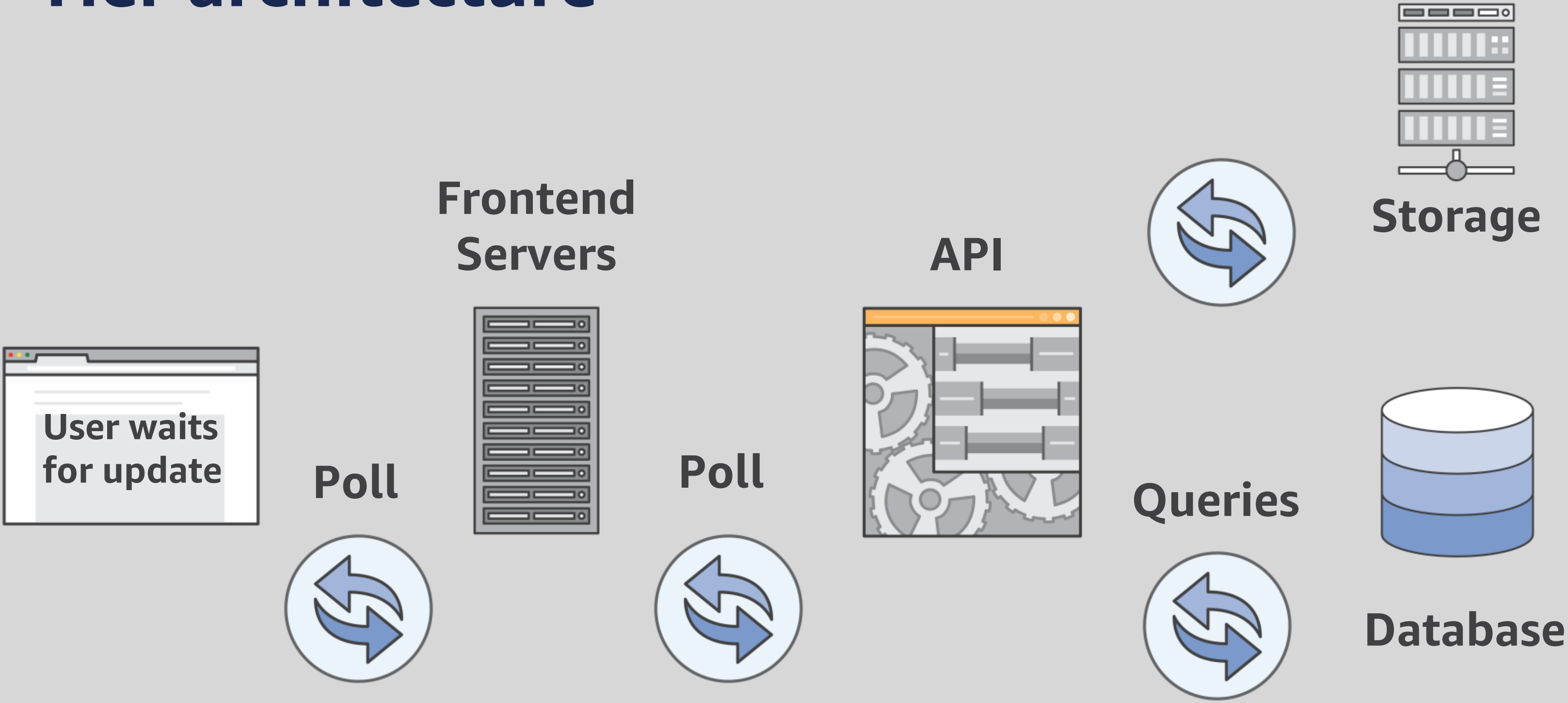
How to build Reactive Architectures on AWS?

Application Architecture

Deployment

What is Reactive Architecture?

N-Tier architecture

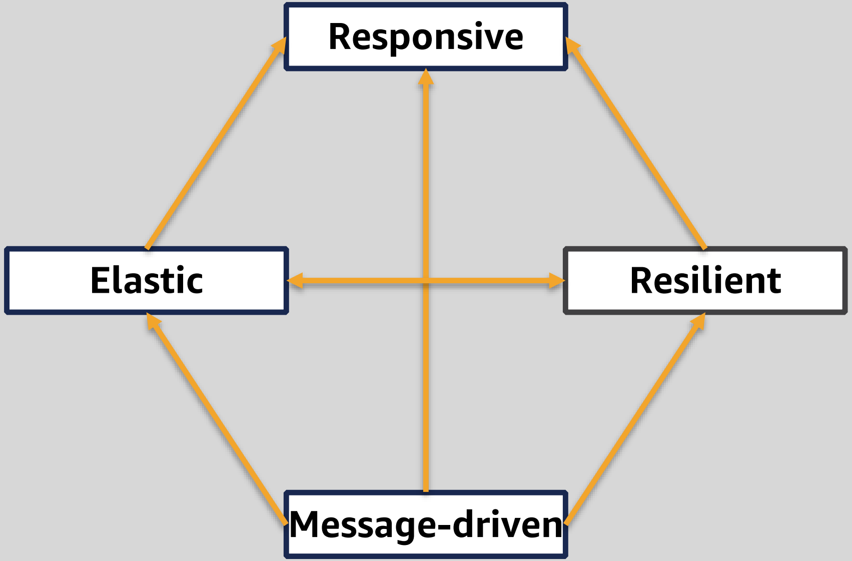


Synchronous

Software moves faster today

Traditional style of applications cannot deliver on these requirements any longer

Reactive Applications



Reactive Architecture

Reactive programming

"A development model focusing on the observation of data streams, reacting on changes, and propagating them."

Reactive system

"An architecture style used to build responsive and robust distributed systems based on asynchronous message-passing."

Reactive Architecture

- Asynchronous message passing
- Non-blocking
    - Higher throughput
    - Efficient compute utilization and lower costs

Reactive Architecture

- Loosely coupled
  - Location independent
  - Easy to extend and maintain
- Push-based

Microservices

Microservices should be stateless.

Keep state in managed services.

No shared libraries or shared SDKs.

Avoid

Host-Affinity.

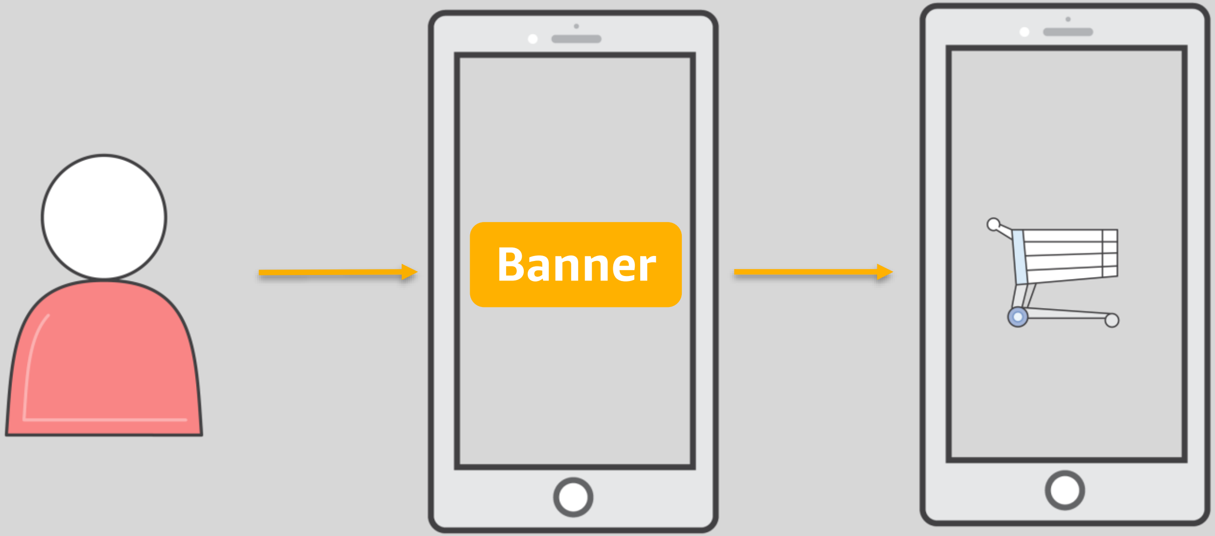
Use lightweight protocols for communication.

Use mechanisms for registration.

aws SUMMIT

How to build Reactive Architectures on AWS?

Example use case



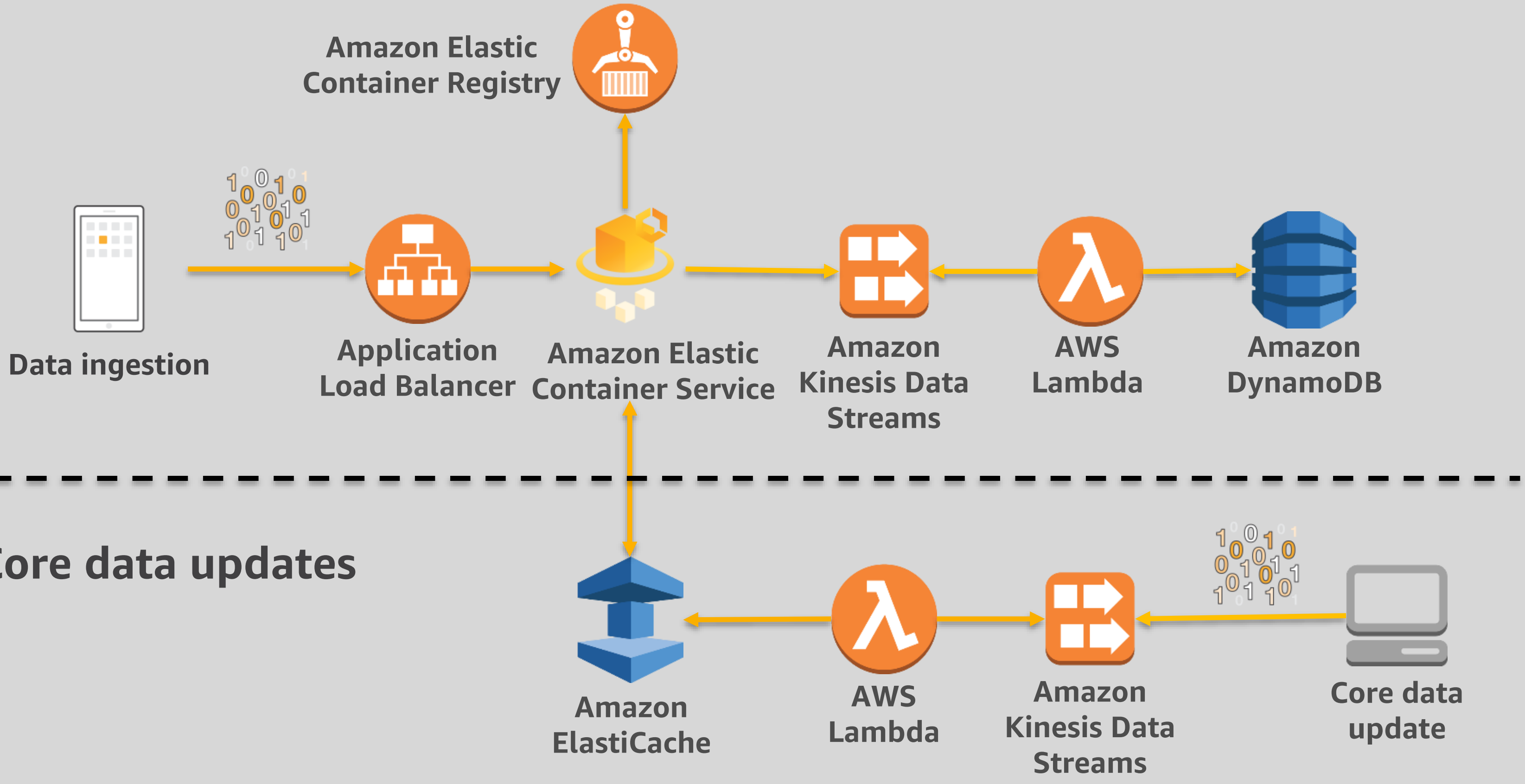
Customer visits website

Customer clicks on a banner

Customer buys from online-shop

High Level Architecture

Data Collection


Amazon Elastic Container Registry
Data ingestion
Application Load Balancer
Amazon Elastic Container Service
Amazon Kinesis Data Streams
AWS Lambda
Amazon DynamoDB
Amazon ElastiCache
AWS Lambda
Amazon Kinesis Data Streams
Core data update

Core data updates

High Level Architecture

- Amazon ECS and Docker used for the main application
- Fargate launch type
- Resiliency and elasticity implemented by using auto scaling

Amazon Elastic Container Service

High Level Architecture

- AWS Lambda functions consume messages
  - Persist data in NoSQL-store
  - Update core-data in Redis
- Send notifications to main application
- Resiliency and scalability part of the service

AWS Lambda

High Level Architecture

- Amazon Kinesis Data Streams used to decouple components
- Asynchronously push event data to NoSQL-store
- Update core-data in Redis

Amazon Kinesis
Data Streams

High Level Architecture

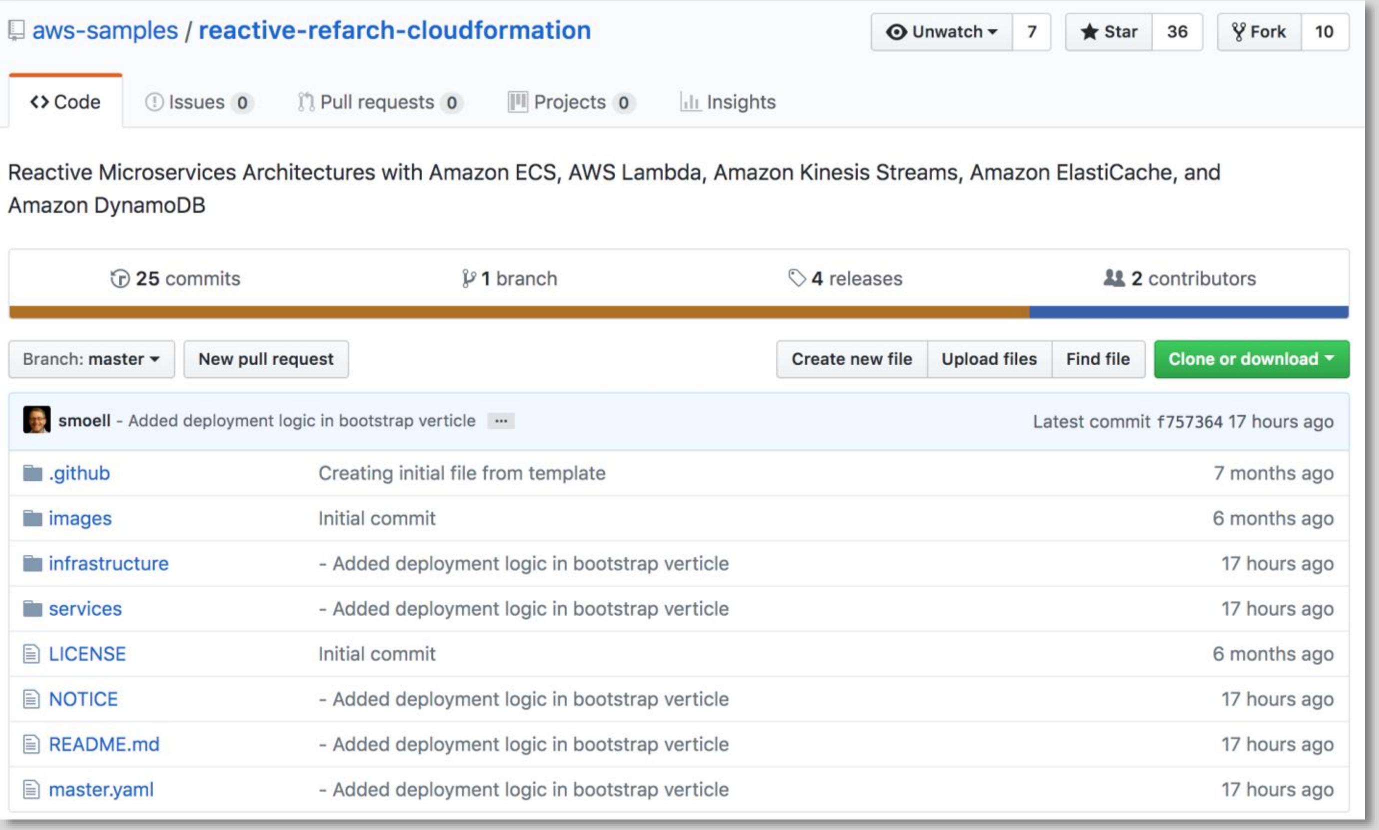
- Amazon ElastiCache with Redis 3 engine
- Multi-AZ setup with failover and one shard
- Used to store core-data
- Notification channel
    - Redis supports pub/sub

Amazon ElastiCache

High Level Architecture

- Amazon DynamoDB NoSQL-store used to persist event-data
- Backup and restore
- Encryption at rest

Amazon DynamoDB



aws-samples / reactive-refarch-cloudformation

Reactive Microservices Architectures with Amazon ECS, AWS Lambda, Amazon Kinesis Streams, Amazon ElastiCache, and Amazon DynamoDB

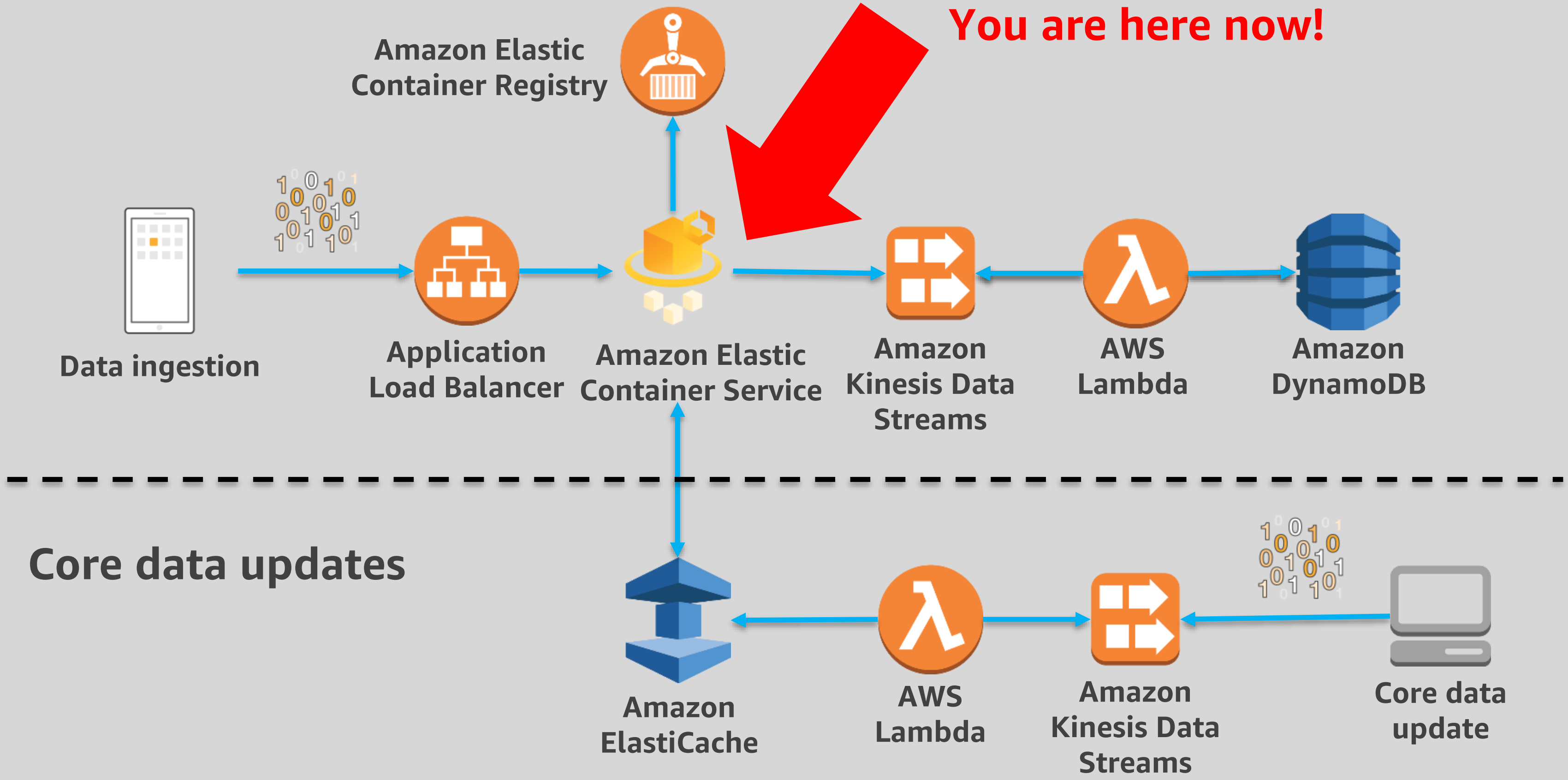
25 commits | 1 branch | 4 releases | 2 contributors

smoell - Added deployment logic in bootstrap verticle ··· Latest commit f757364 17 hours ago

.github	Creating initial file from template	7 months ago
images	Initial commit	6 months ago
infrastructure	- Added deployment logic in bootstrap verticle	17 hours ago
services	- Added deployment logic in bootstrap verticle	17 hours ago
LICENSE	Initial commit	6 months ago
NOTICE	- Added deployment logic in bootstrap verticle	17 hours ago
README.md	- Added deployment logic in bootstrap verticle	17 hours ago
master.yaml	- Added deployment logic in bootstrap verticle	17 hours ago

Application Architecture

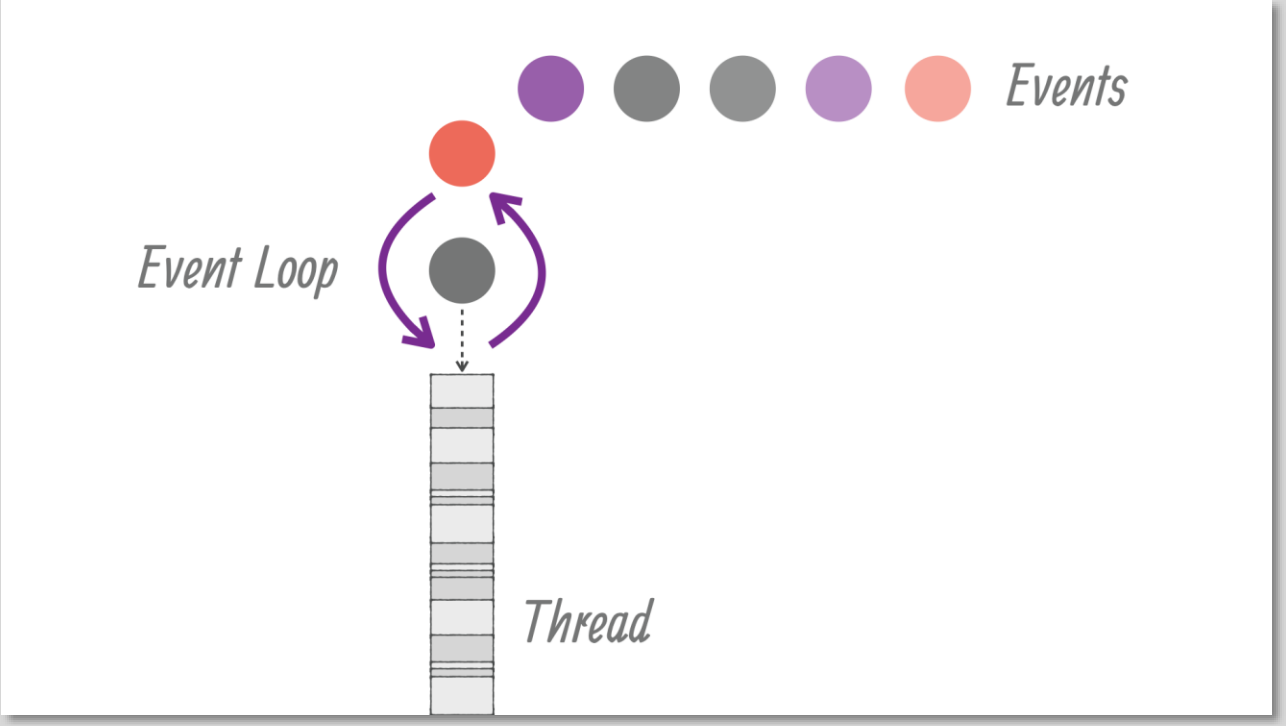
Data Collection


You are here now!
Amazon Elastic Container Registry
Data ingestion
Application Load Balancer
Amazon Elastic Container Service
Amazon Kinesis Data Streams
AWS Lambda
Amazon DynamoDB
Amazon ElastiCache
AWS Lambda
Amazon Kinesis Data Streams
Core data update

Core data updates

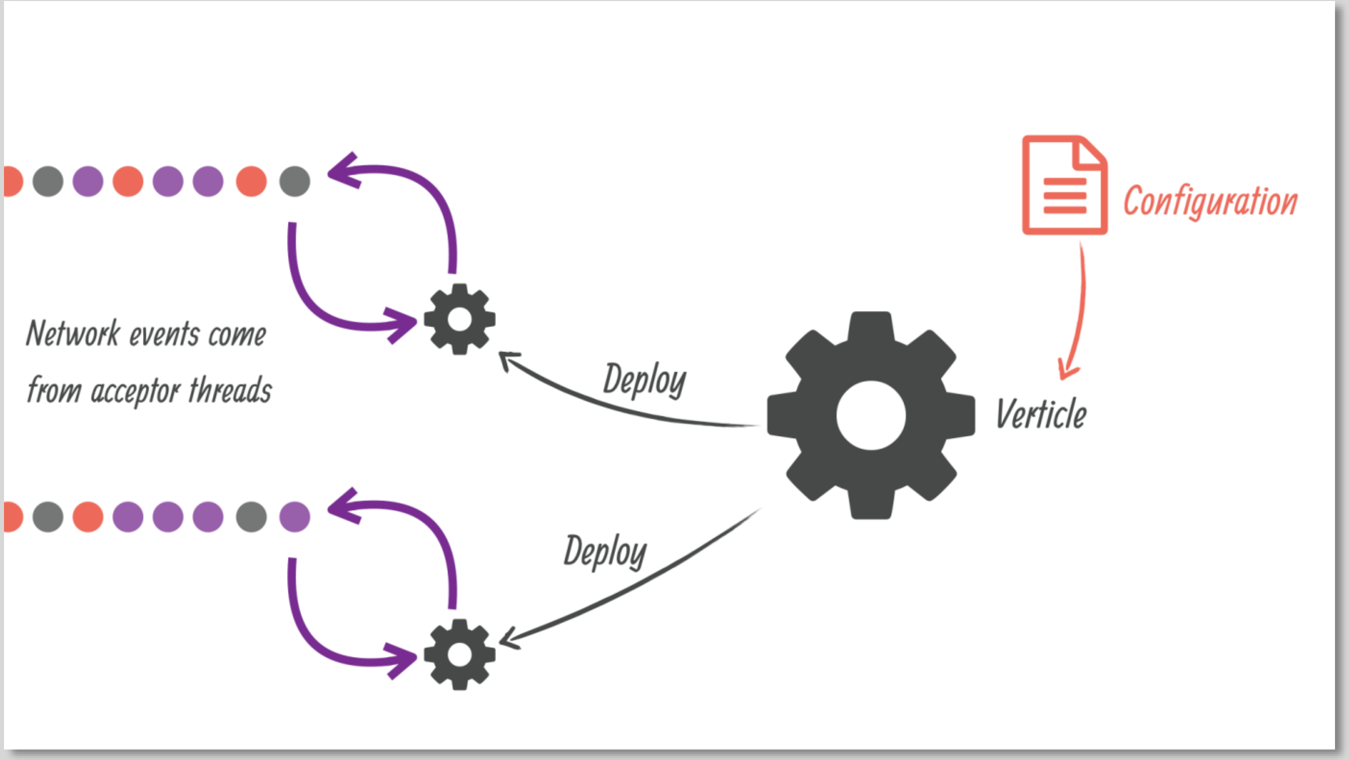
Application Architecture

Application Architecture



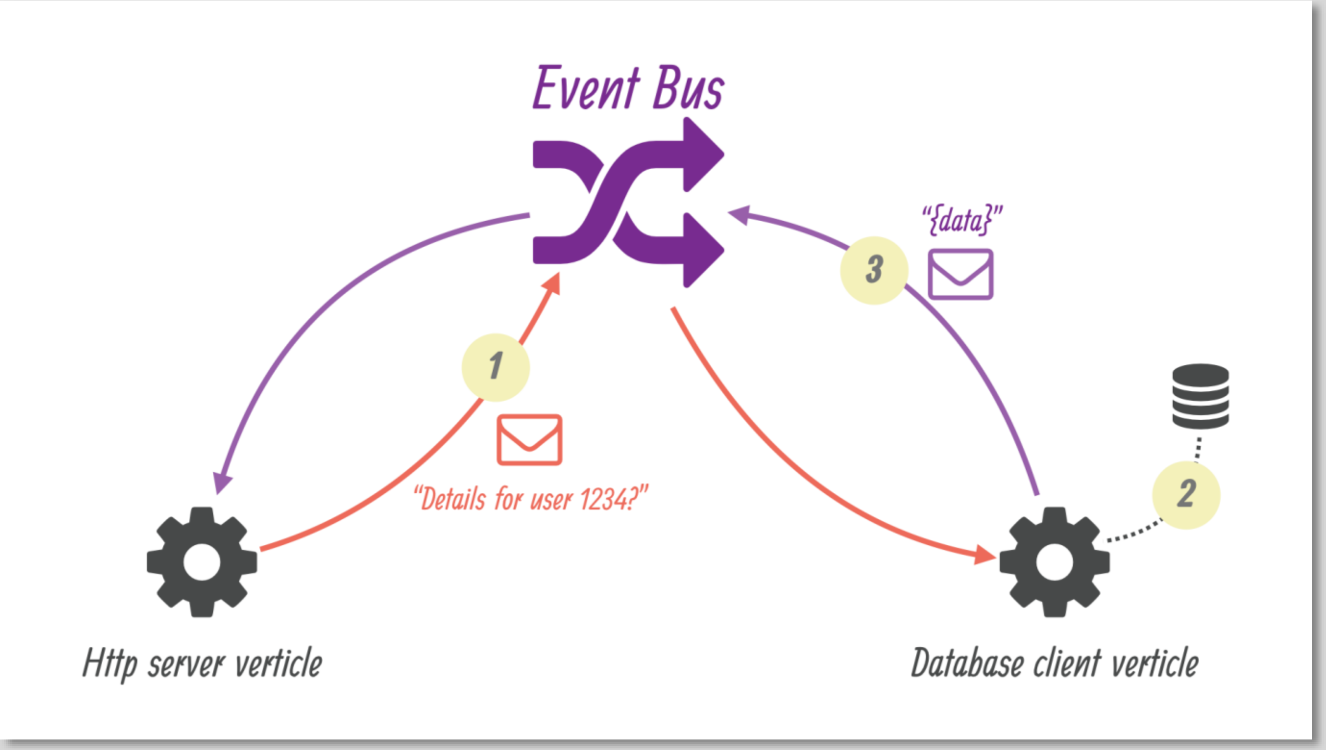
Source: http://vertx.io/docs/guide-for-java-devs/

Application Architecture



Source: http://vertx.io/docs/guide-for-java-devs/

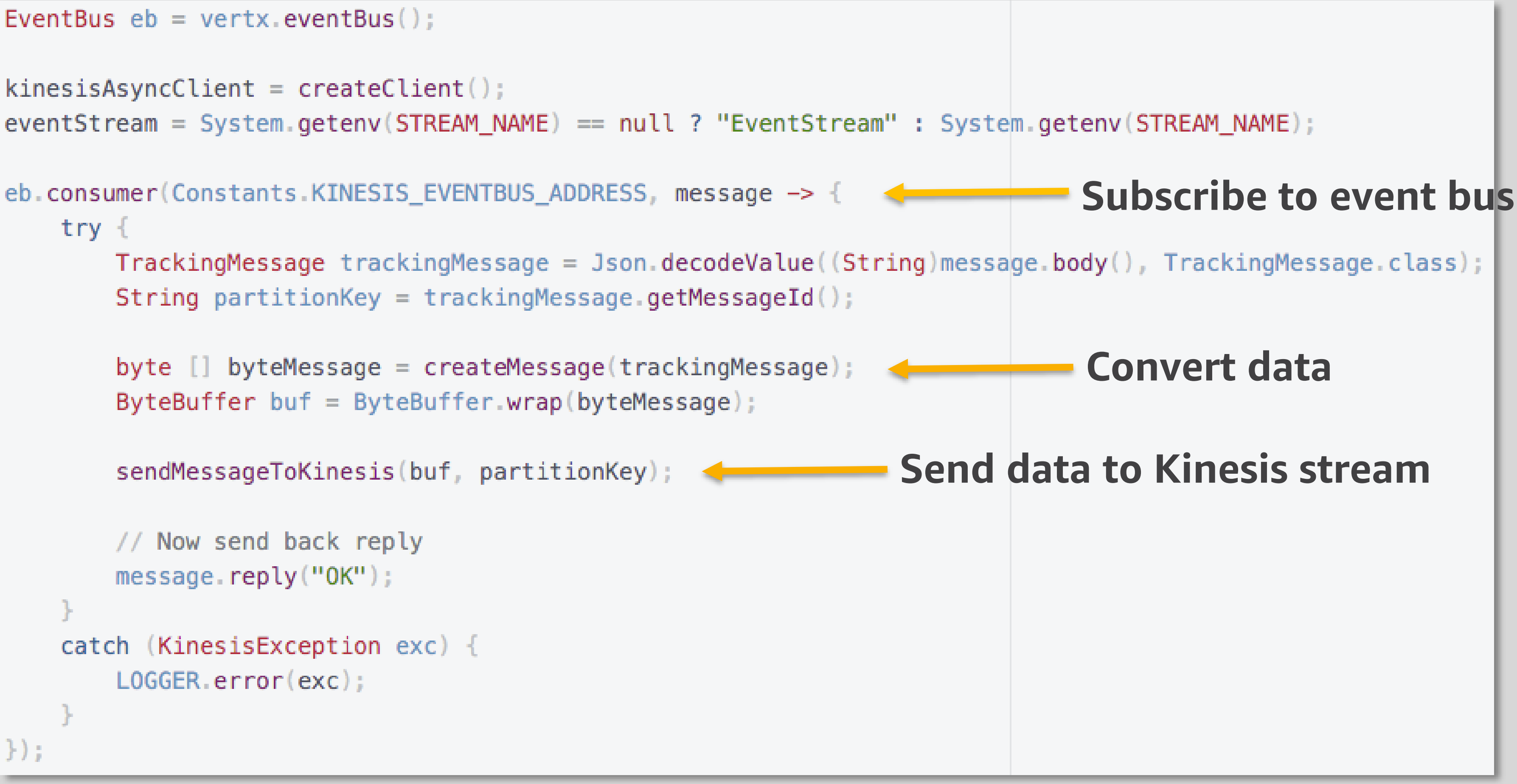
Application Architecture



Source: http://vertx.io/docs/guide-for-java-devs/

Application Architecture

- *HttpVerticle*: exposes HTTP endpoint
- *CacheVerticle*: implements L1 cache
- *RedisVerticle*: implements Redis access
- *KinesisVerticle*: messages to Amazon Kinesis Data Stream



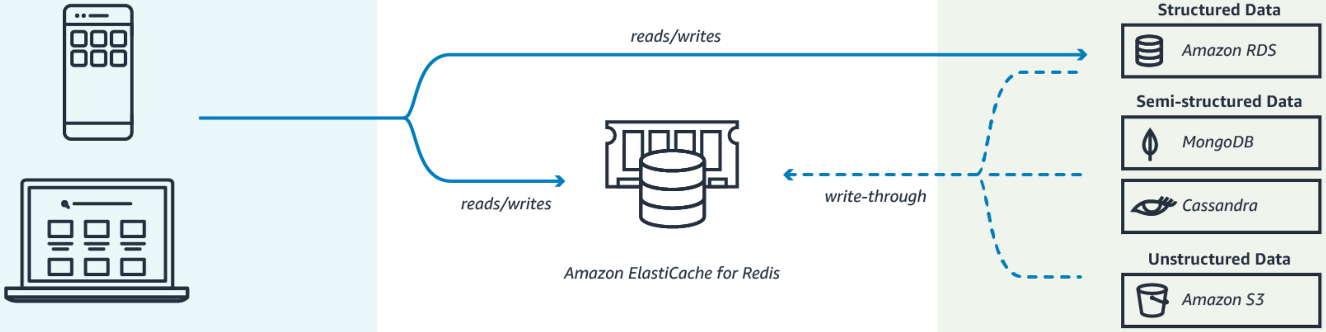
```
EventBus eb = vertx.eventBus();

kinesisAsyncClient = createClient();
eventStream = System.getenv(STREAM_NAME) == null ? "EventStream" : System.getenv(STREAM_NAME);

eb.consumer(Constants.KINESIS_EVENTBUS_ADDRESS, message -> {
    try {
        TrackingMessage trackingMessage = Json.decodeValue((String)message.body(), TrackingMessage.class);
        String partitionKey = trackingMessage.getMessageId();

        byte [] byteMessage = createMessage(trackingMessage);
        ByteBuffer buf = ByteBuffer.wrap(byteMessage);

        sendMessageToKinesis(buf, partitionKey);

        // Now send back reply
        message.reply("OK");
    }
    catch (KinesisException exc) {
        LOGGER.error(exc);
    }
});
```

Subscribe to event bus

Convert data

Send data to Kinesis stream

Application Architecture



reads/writes

reads/writes

Amazon ElastiCache for Redis

write-through

Structured Data

Amazon RDS

Semi-structured Data

MongoDB

Cassandra

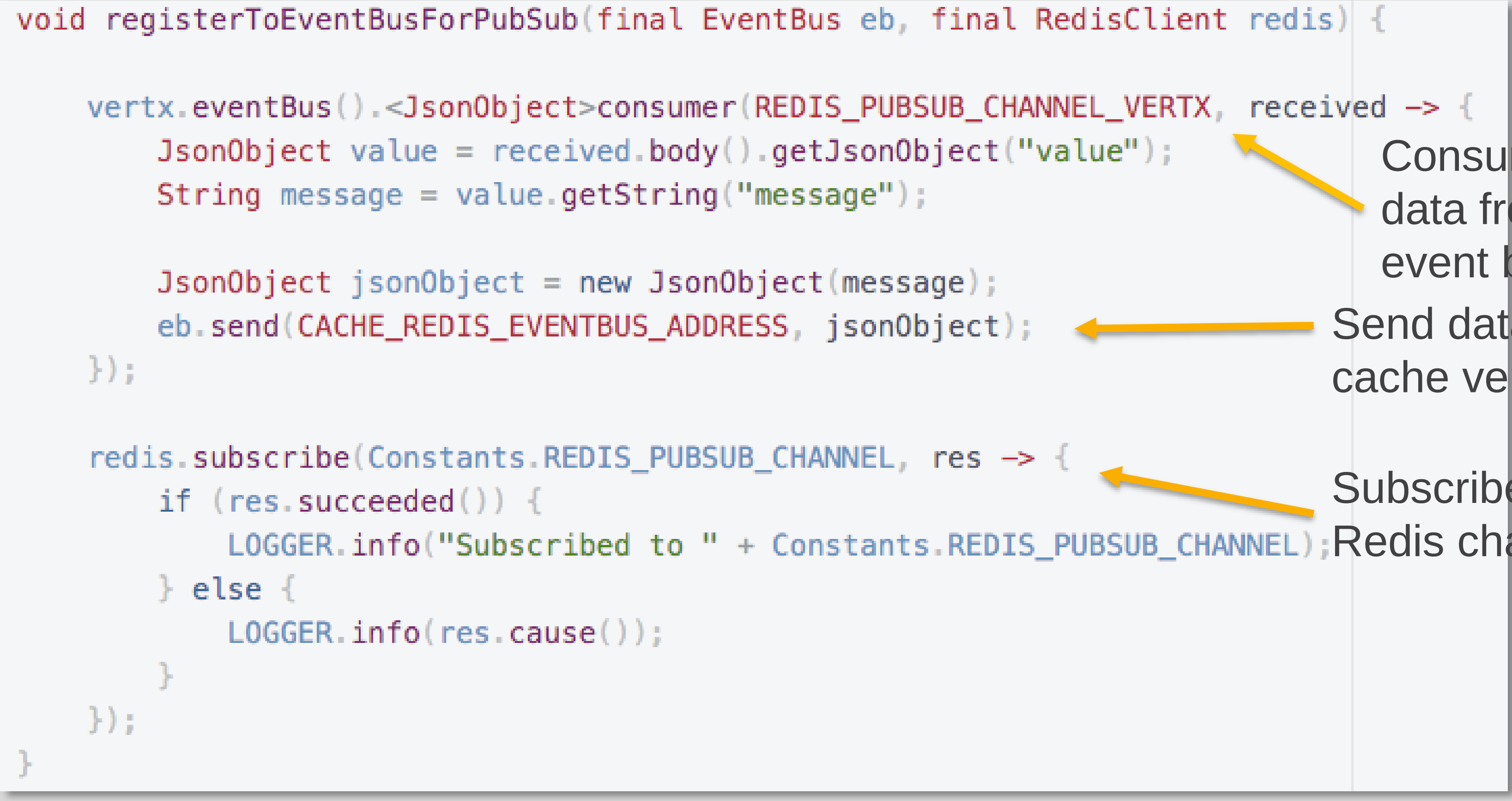
Unstructured Data

Amazon S3

Clients

In-memory data store and cache to decrease access latency, increase throughput, and ease the load off databases and applications.

Databases and File Stores



```
void registerToEventBusForPubSub(final EventBus eb, final RedisClient redis) {

    vertx.eventBus().<JsonObject>consumer(REDIS_PUBSUB_CHANNEL_VERTX, received -> {
        JsonObject value = received.body().getJsonObject("value");
        String message = value.getString("message");

        JsonObject jsonObject = new JsonObject(message);
        eb.send(CACHE_REDIS_EVENTBUS_ADDRESS, jsonObject);
    });

    redis.subscribe(Constants.REDIS_PUBSUB_CHANNEL, res -> {
        if (res.succeeded()) {
            LOGGER.info("Subscribed to " + Constants.REDIS_PUBSUB_CHANNEL);
        } else {
            LOGGER.info(res.cause());
        }
    });
}
```

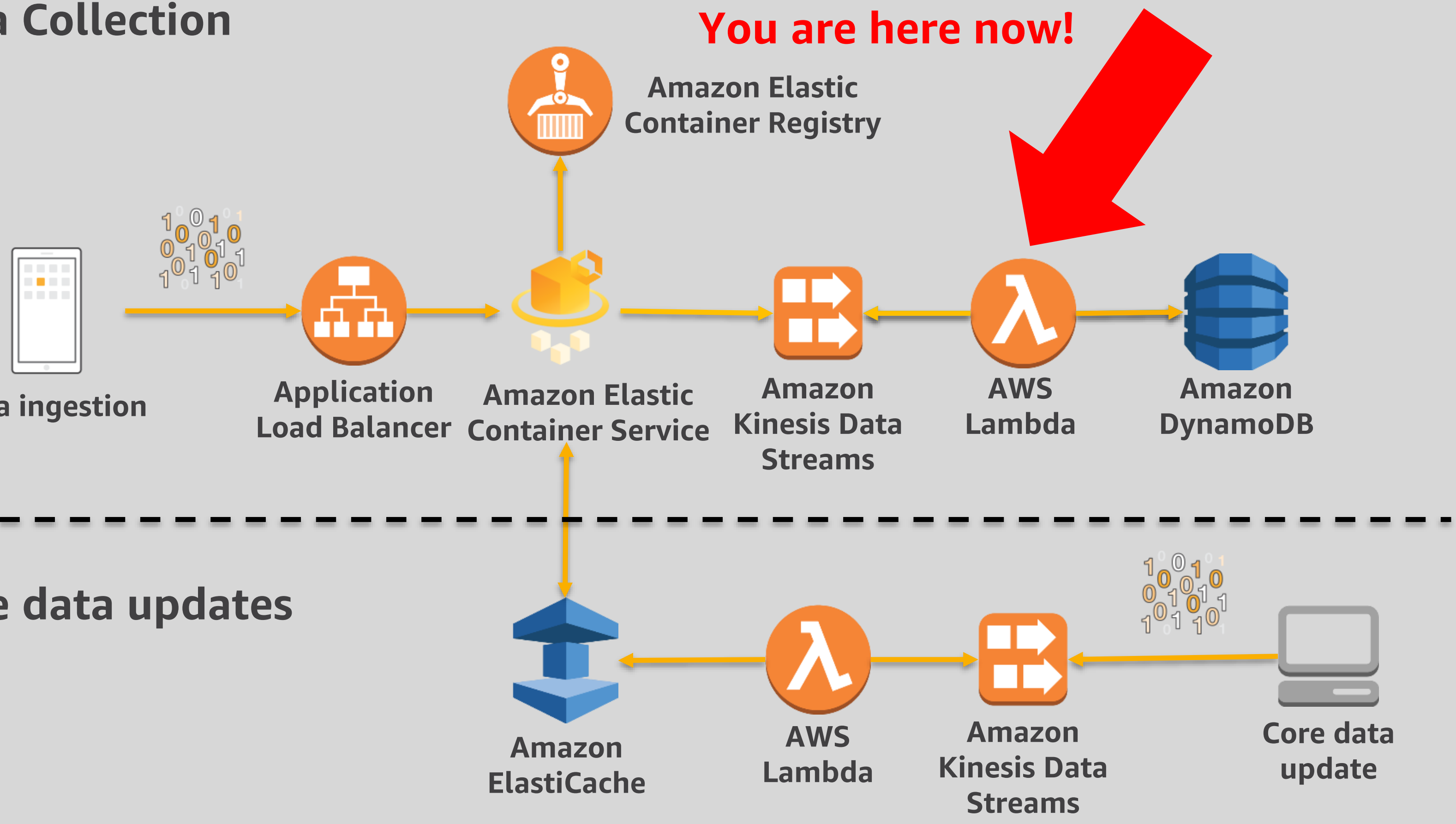
Consume data from event bus

Send data to cache verticle

Subscribe to Redis channel

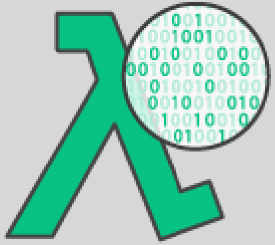
Data Collection

You are here now!



Core data updates

Application Architecture

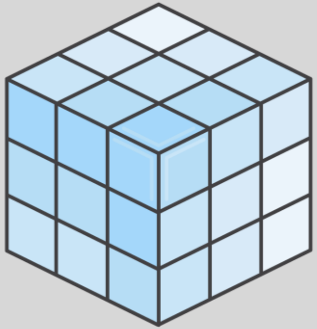


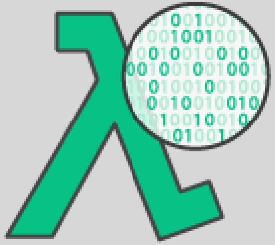
AWS Lambda

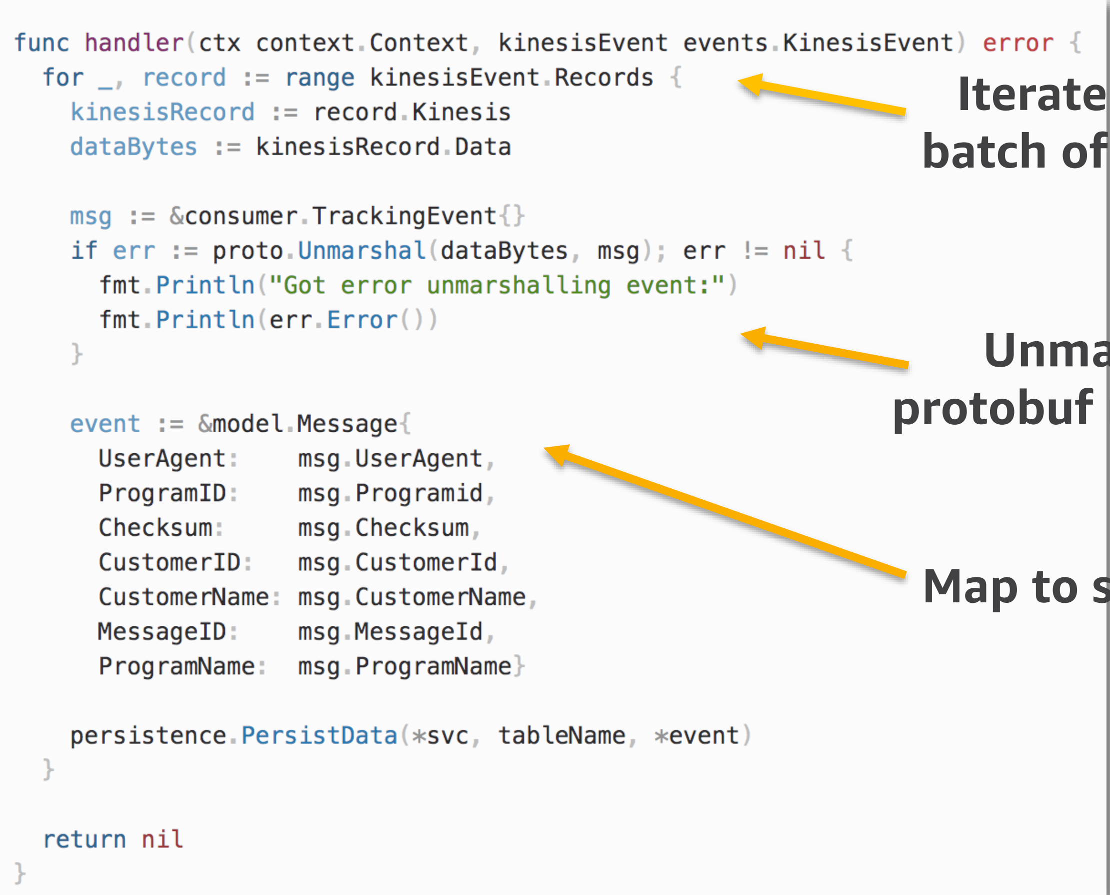
Lambda considerations and best practices

AWS Lambda is stateless—architect accordingly

- Assume no affinity with underlying compute infrastructure
- Local filesystem access and child process may not extend beyond the lifetime of the Lambda request







```go
func handler(ctx context.Context, kinesisEvent events.KinesisEvent) error {
  for _, record := range kinesisEvent.Records {
    kinesisRecord := record.Kinesis
    dataBytes := kinesisRecord.Data

    msg := &consumer.TrackingEvent{}
    if err := proto.Unmarshal(dataBytes, msg); err != nil {
      fmt.Println("Got error unmarshalling event:")
      fmt.Println(err.Error())
    }

    event := &model.Message{
      UserAgent:    msg.UserAgent,
      ProgramID:    msg.Programid,
      Checksum:     msg.Checksum,
      CustomerID:   msg.CustomerId,
      CustomerName: msg.CustomerName,
      MessageID:    msg.MessageId,
      ProgramName:  msg.ProgramName}

    persistence.PersistData(*svc, tableName, *event)
  }

  return nil
}
```

Iterate over batch of events

Unmarshal protobuf messages

Map to struct

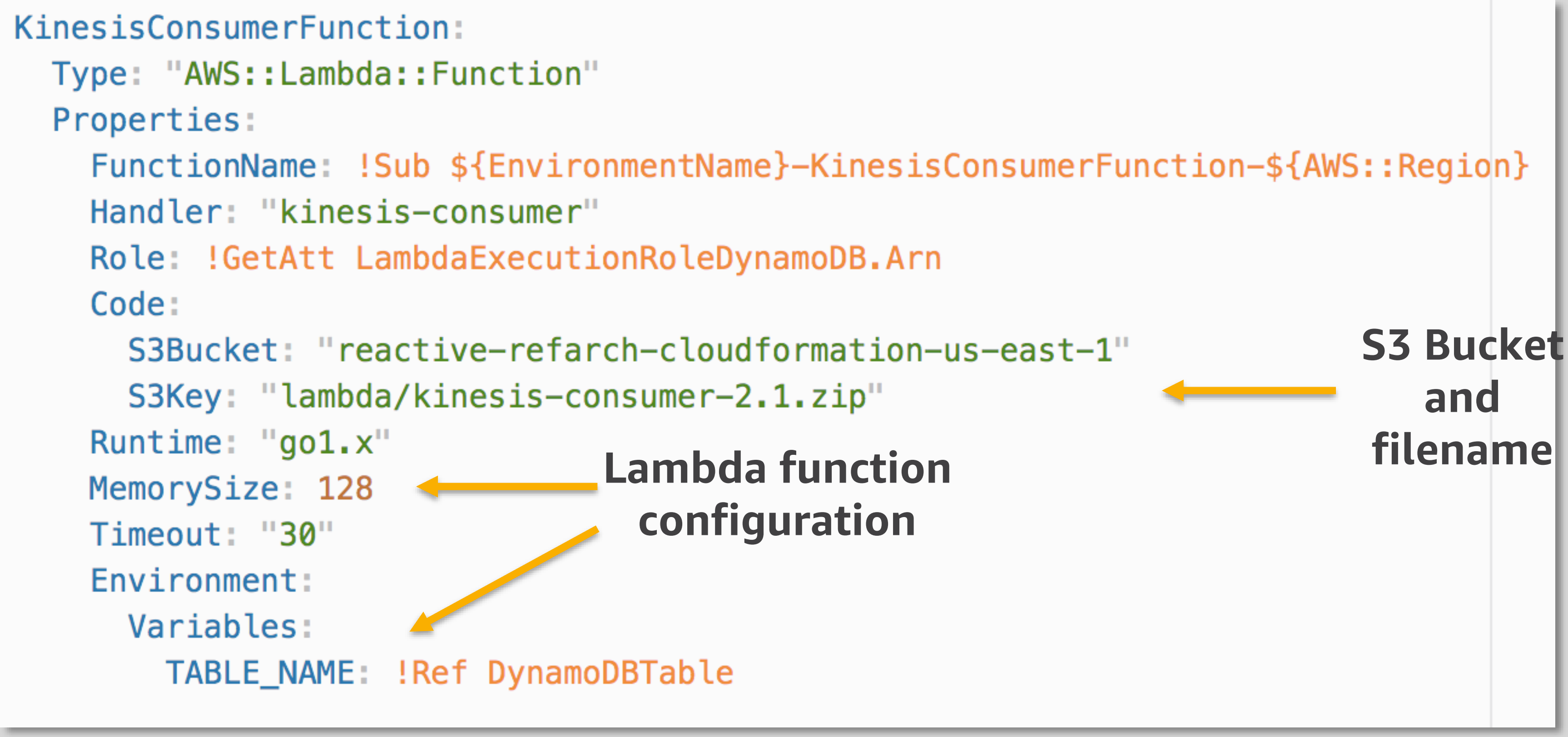
Design principles

- Use push instead of pull
- Avoid blocking calls
- Decouple your services using async message passing
- Keep state in managed services
- Use caching

Deployment

Deployment

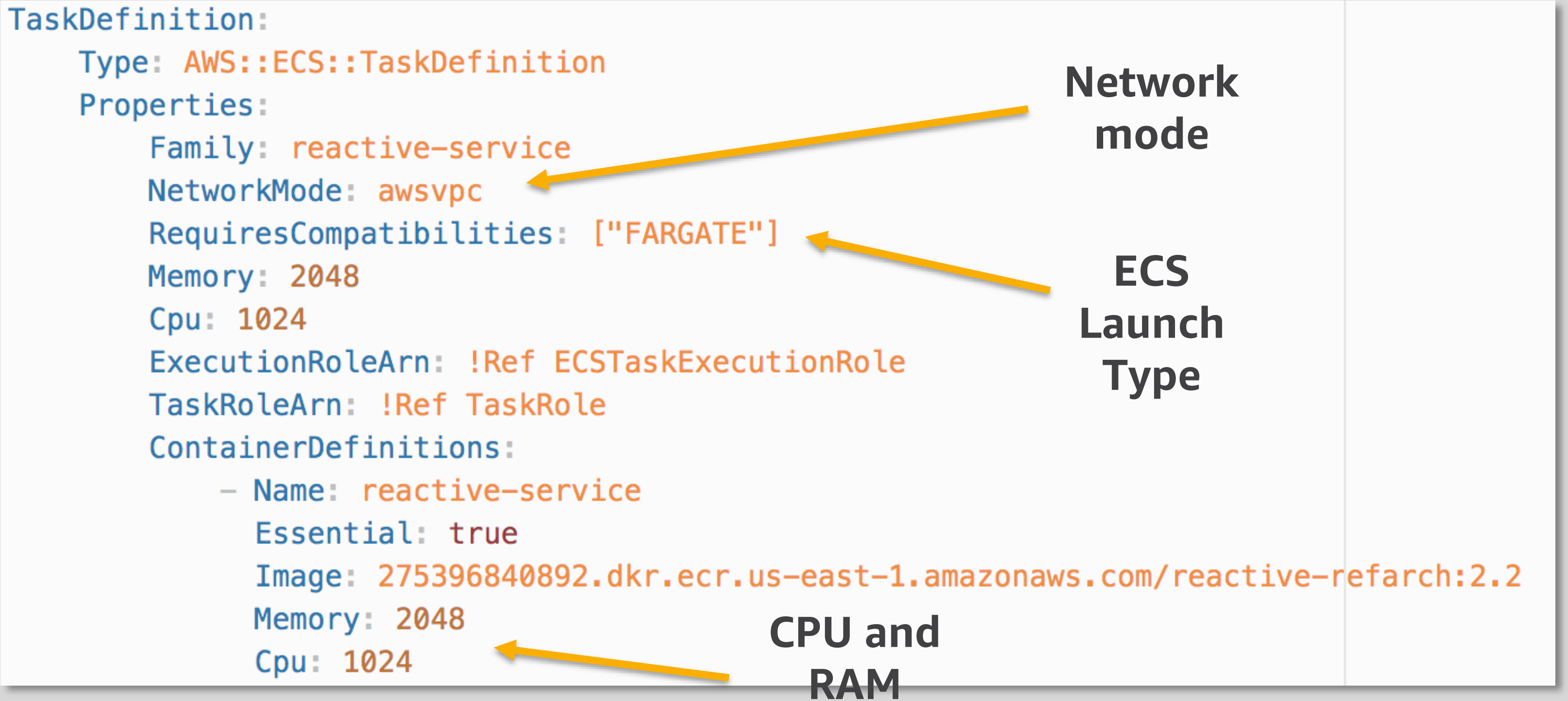
AWS CloudFormation



```
KinesisConsumerFunction:
  Type: "AWS::Lambda::Function"
  Properties:
    FunctionName: !Sub ${EnvironmentName}-KinesisConsumerFunction-${AWS::Region}
    Handler: "kinesis-consumer"
    Role: !GetAtt LambdaExecutionRoleDynamoDB.Arn
    Code:
      S3Bucket: "reactive-refarch-cloudformation-us-east-1"
      S3Key: "lambda/kinesis-consumer-2.1.zip"
    Runtime: "go1.x"
    MemorySize: 128
    Timeout: "30"
    Environment:
      Variables:
        TABLE_NAME: !Ref DynamoDBTable
```

S3 Bucket and filename

Lambda function configuration



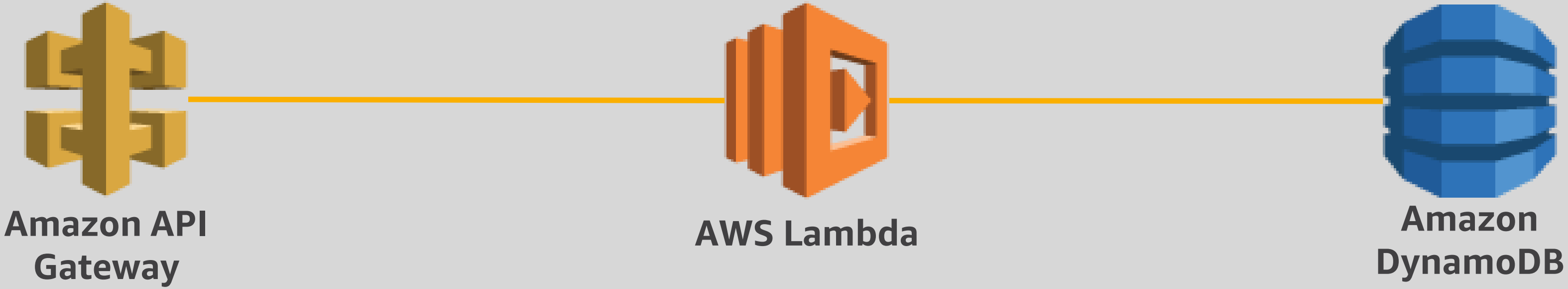
```
TaskDefinition:
    Type: AWS::ECS::TaskDefinition
    Properties:
        Family: reactive-service
        NetworkMode: awsvpc
        RequiresCompatibilities: ["FARGATE"]
        Memory: 2048
        Cpu: 1024
        ExecutionRoleArn: !Ref ECSTaskExecutionRole
        TaskRoleArn: !Ref TaskRole
        ContainerDefinitions:
            - Name: reactive-service
              Essential: true
              Image: 275396840892.dkr.ecr.us-east-1.amazonaws.com/reactive-refarch:2.2
              Memory: 2048
              Cpu: 1024
```

Network mode

ECS Launch Type

CPU and RAM

Go build something!



Amazon API Gateway

AWS Lambda

Amazon DynamoDB

Related Sessions

Container-based Architectures on AWS (Blackfoot)

Kubernetes Running on AWS (Blackfoot)

Serverless Architectural Patterns (Amelia)

Deep Dive into Concepts and Tools for Analyzing Streaming Data on AWS (Coral)

Question? Ask this guy at the Ask an Architect booth:

Please complete the session survey in the summit mobile app.

Thank you!