Event-Driven Architecture and Serverless

Kyle Le
Solution Architect

Dan Shays
Sr. Solution Architect

John Ringhofer
Sr. Solution Architect

Red Hat

Agenda

- Re-cap Container and Kubernetes
- Traditional Architecture Pain Points
- What is Serverless?
- Why Serverless?
- Running apps serverless-ly
- A Solution Appears
- Demo Walkthrough
- Next Steps

Red Hat

Source: Dilbert, by Scott Adams

Red Hat

Containers and Kubernetes: A Primer

Red Hat

An application, decomposed

For an application to run, it needs…

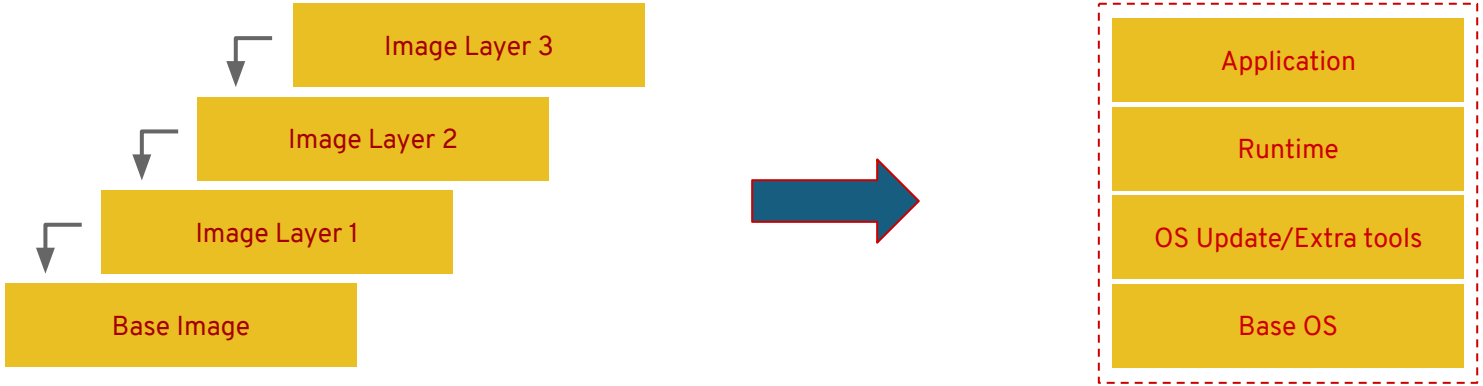
Compiled and built application code binary
Dependencies to make the application code binary work
Dependencies to make the middleware work
Middleware
Operating System
Hardware

A Container is…

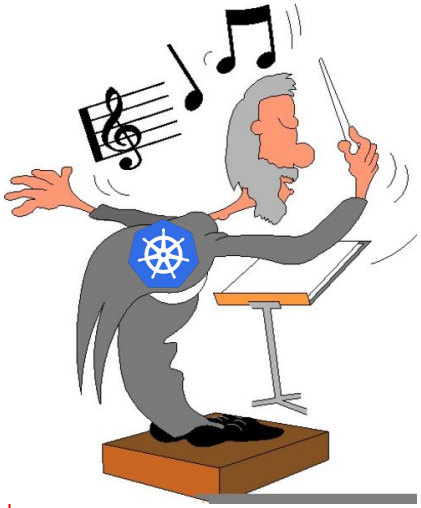
Containers bundle all of *this* into **one standardized deliverable**:

Compiled and built application code binary
Dependencies to make the application code binary work
Dependencies to make the middleware work
Middleware
Operating System
Hardware

...built in layers, like this!



Kubernetes - What is?



Kubernetes is open source container *orchestration* - it automates deployment, scaling, and management of containers.

Red Hat

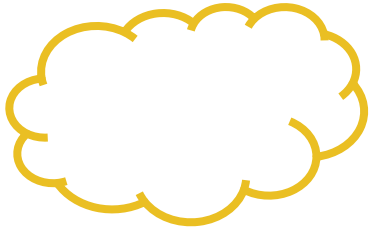
Traditional Architecture Pain Points

Red Hat

Application Architecture

Choices exist for how to build and deploy an application:

Monolith



Cloud Native

Serverless

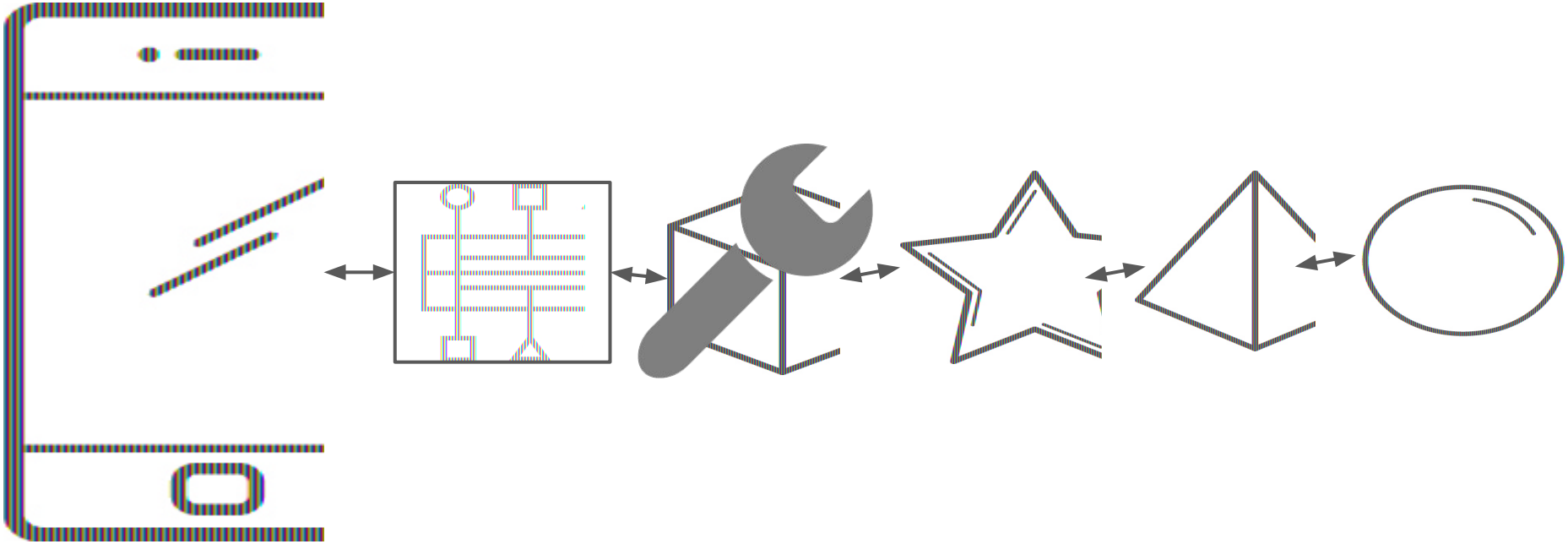
Architecture Pain Points

1. Tight Coupling
2. Baton Dropping
3. Call Chain Latency...ing
4. Sizing/Scaling

Tight Coupling

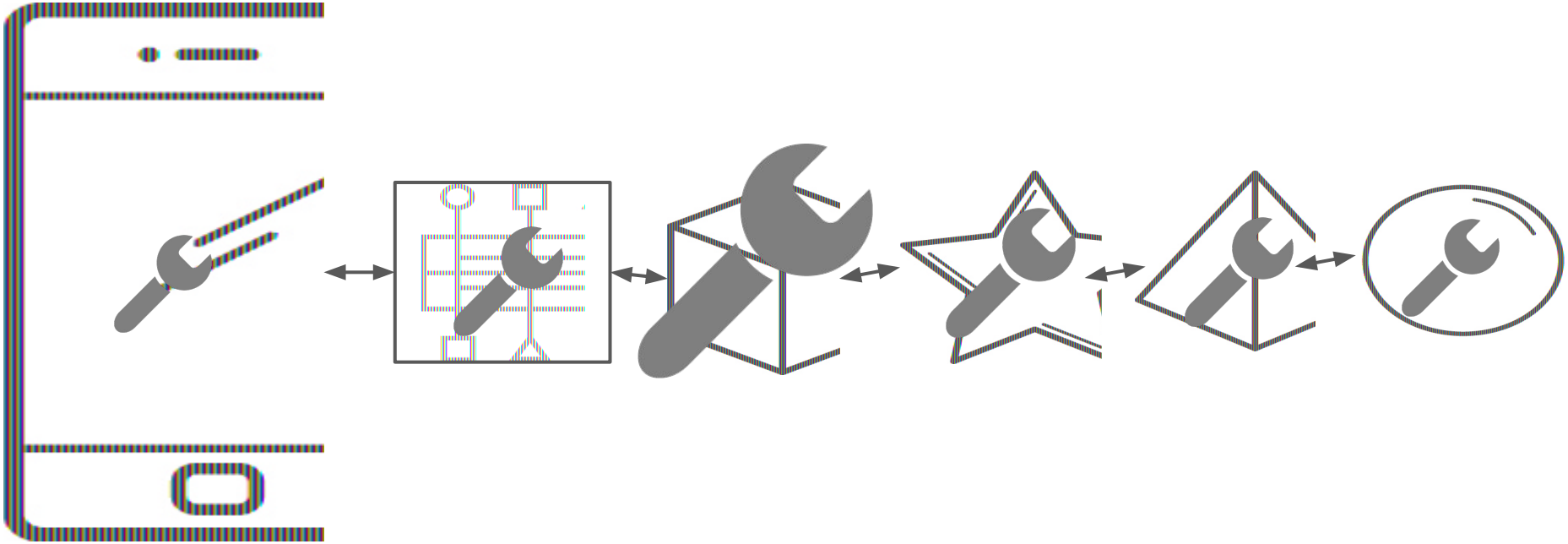
When one thing changes, *everything* has to (potentially) change.

Tight Coupling



Red Hat

Tight Coupling

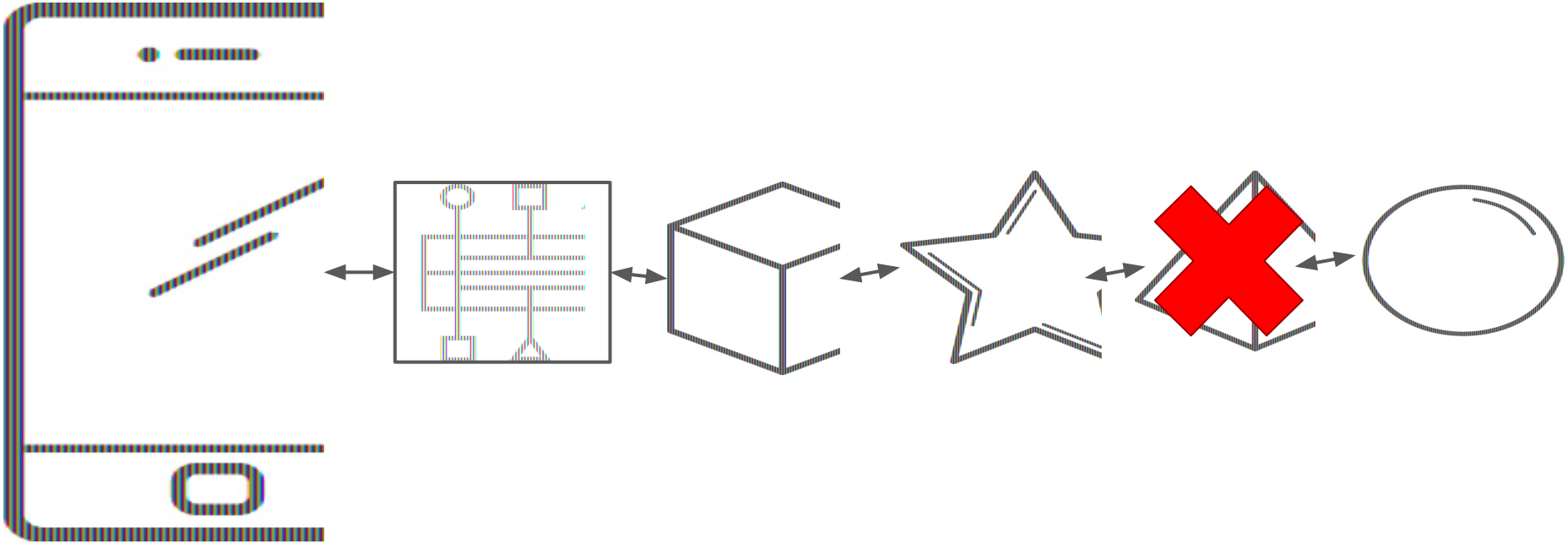


Red Hat

Baton Dropping

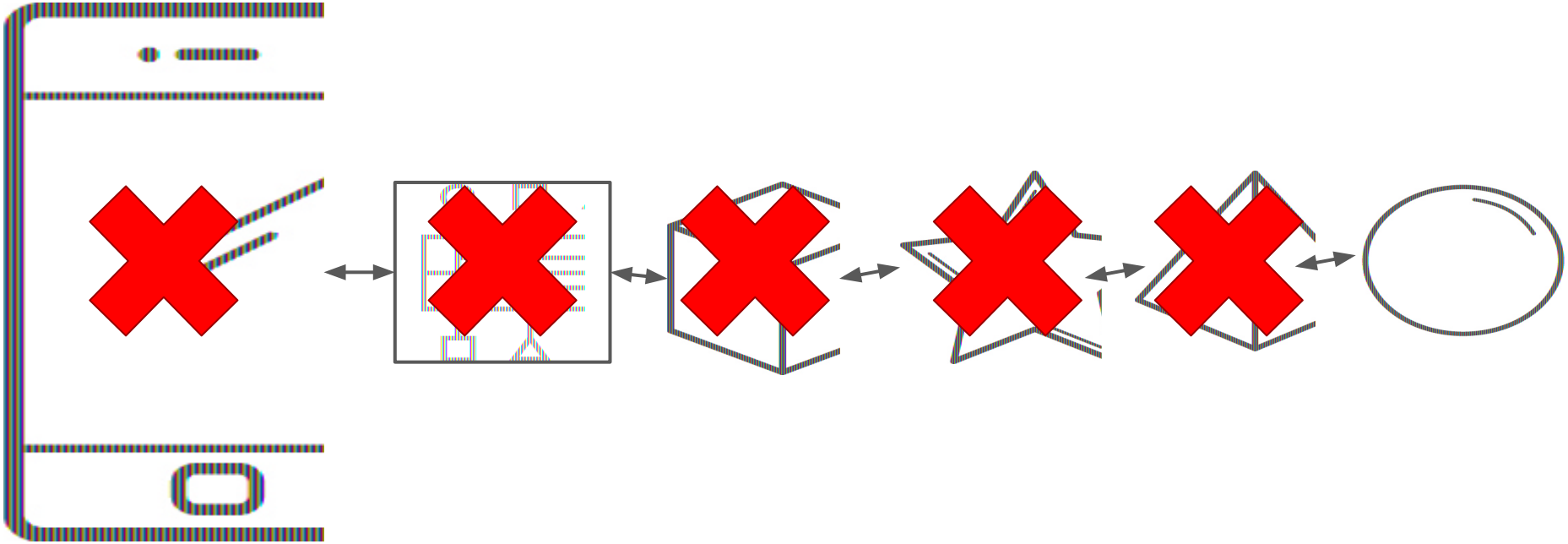
When something breaks, *everything* breaks.

Baton Dropping



Red Hat

Baton Dropping



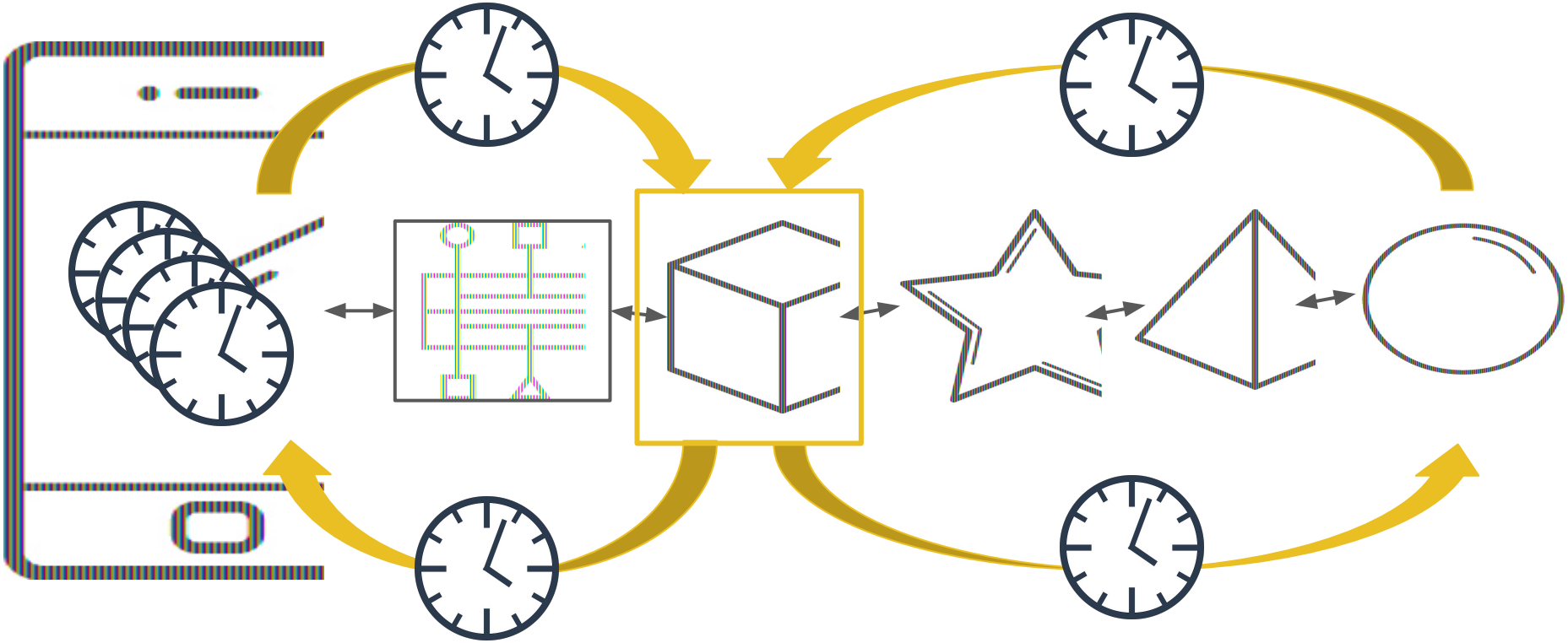
Red Hat

Call Chain Latency...ing

- Every call in the chain takes time to execute the code.
- Every hop between calls takes network time.
- Jumping between data centers and clouds add MORE time/latency.

Red Hat

Call Chain Latency...ing



Red Hat

Sizing/Scaling

How many instances of the running application should we have? 1? 2? 5?

It's expensive to guess too many (because of wasted infrastructure!) and even more expensive to guess too few - because customers expect responsiveness.

So...again, the pain points are:

1. Tight Coupling
2. Baton Dropping
3. Call Chain Latency...ing
4. Sizing/Scaling

Effects of Microservices/Clouds

1. *Longer* call chains
2. *More* network latency
3. Splitting call chains
4. Scaling is even harder - so many services!

Serverless: What is it, exactly?

Red Hat

Serverless, Defined

Serverless computing is...

- a cloud computing **execution model**
- where the applications are written by the cloud consumer
- and infrastructure is managed by the cloud provider

"Infrastructure": from hardware all the way up the stack to the number of instances of applications running

Red Hat

Serverless vs FaaS

Functions as a Service (FaaS): small pieces of code that you trust someone else to run for you.

In general, architecture options are limited to what they (the “someone else”) run, and how they choose to run it.

Serverless vs FaaS

Serverless *isn't* the same as FaaS – but FaaS systems **do** generally use the Serverless execution model to manage how they run.

Serverless TL;DR

Serverless uses the power of **containers** and **automation** to minimize the thought and work needed to run applications.

Why Serverless?

Red Hat

Okay, but *why* serverless?

"Running containers is already simpler than running VMs!"

or

"Just getting to containers is so huge for us, let's look at that first!"

So...what benefit does this *new* thing bring?

Red Hat

~~Market Trends~~ Why You Care

TL;DR: *"Use Serverless to optimize the benefits of the cloud."* [2]

40%

of enterprises adopted Serverless technologies or practices with expected growth coming in the next 12 to 18 months.[1]

Vendor lock-in is the second biggest concern when adopting Serverless technologies.[1]

60%

of the serverless practitioners reported *"reduction of operational costs"* with the second biggest benefit being *"scale with demand automatically"*

Source:

1. https://www.oreilly.com/radar/oreilly-serverless-survey-2019-concerns-what-works-and-what-to-expect/
2. Forrester - Now Tech: Serverless, Q4 2019

Red Hat

Containerized Applications

Containerized applications typically run within a pod. Pods can be scaled to run multiple versions of the application (*super* cool), but auto-scaling based on any particular factor is tricky - even if the application could handle it just fine.

Red Hat

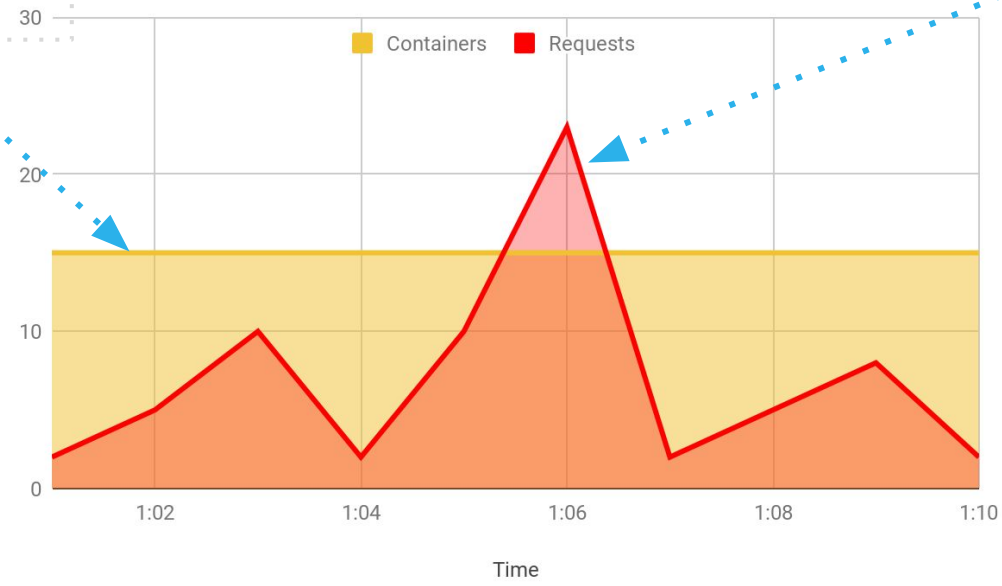
Containerized Applications

Applications deployed serverless-ly, on the other hand, scale up and then back down based on...something - there are multiple options.

Infrastructure without Serverless

Under provisioning
Slow, or non-responsive.
Lost business revenue,
poor quality of service.

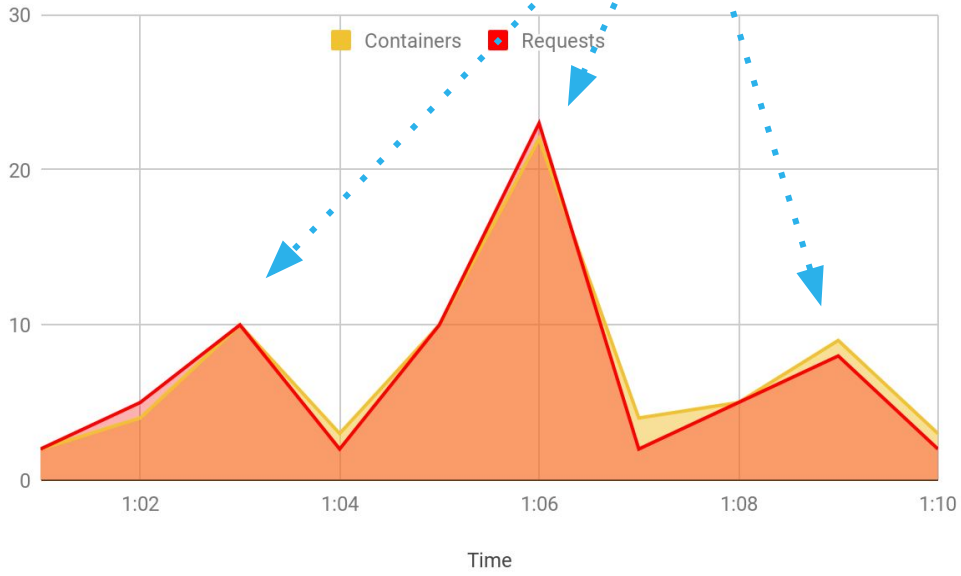
Over provisioning
Lost time in inaccurate
capacity planning, IT cost of
idle resources.



Red Hat

Serverless is Cool

Infrastructure more closely matches demand.
Applications are responsive.
Infra costs have less idle.



Red Hat

A Solution Appears!

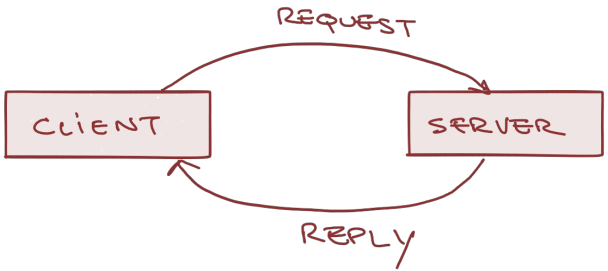
Red Hat

A Solution Appears!

1. Asynchronous Processing
2. Event-Driven Architecture
3. Serverless Deployment ✅

Red Hat

Request/Reply vs Event-Driven



Synchronous
Ephemeral
Low composability
Simplified model
Low tolerance to failure
Best practices evolved as REST

Asynchronous
Optionally Persistent
Highly composable
Complex model
High tolerance to failure
Best practices still evolving
Decoupled

Red Hat

A Solution Appears!

1. Asynchronous Processing ✅
2. Event-Driven Architecture
3. Serverless Deployment ✅

Red Hat

All About Events

"Event" - an action or occurrence that happened in the past as a result of something (usually an end user, could also be another system) interacting with a system.

Like...

- placing an order
- opening a new account
- making an insurance claim

Red Hat

Characteristics of an Event

- Immutable
- Shareable
- Can be persisted

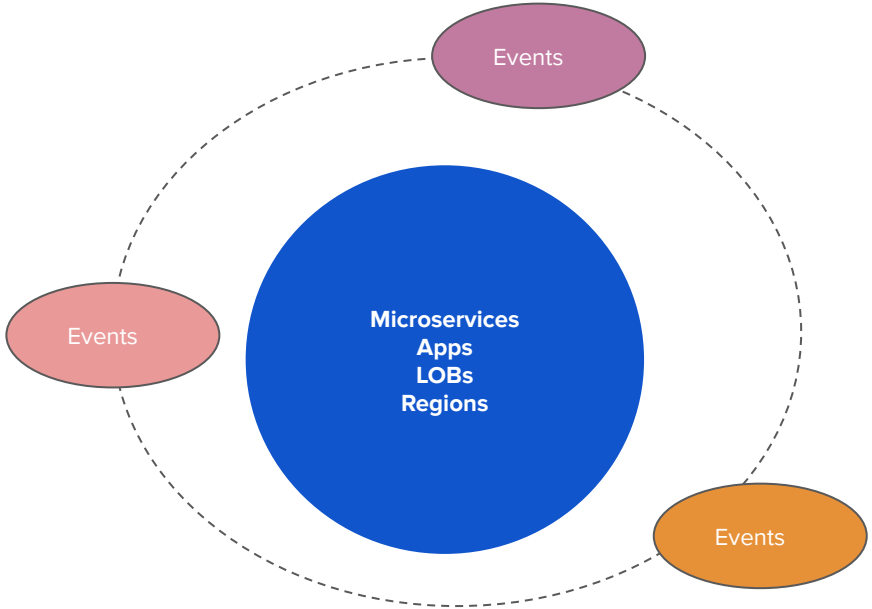
Event Types

- Notification
- State Transfer (command)
- Event sourcing/CQRS

https://martinfowler.com/articles/201701-event-driven.html

Red Hat

A Change in Perspective



System-centric, and **data**-centric

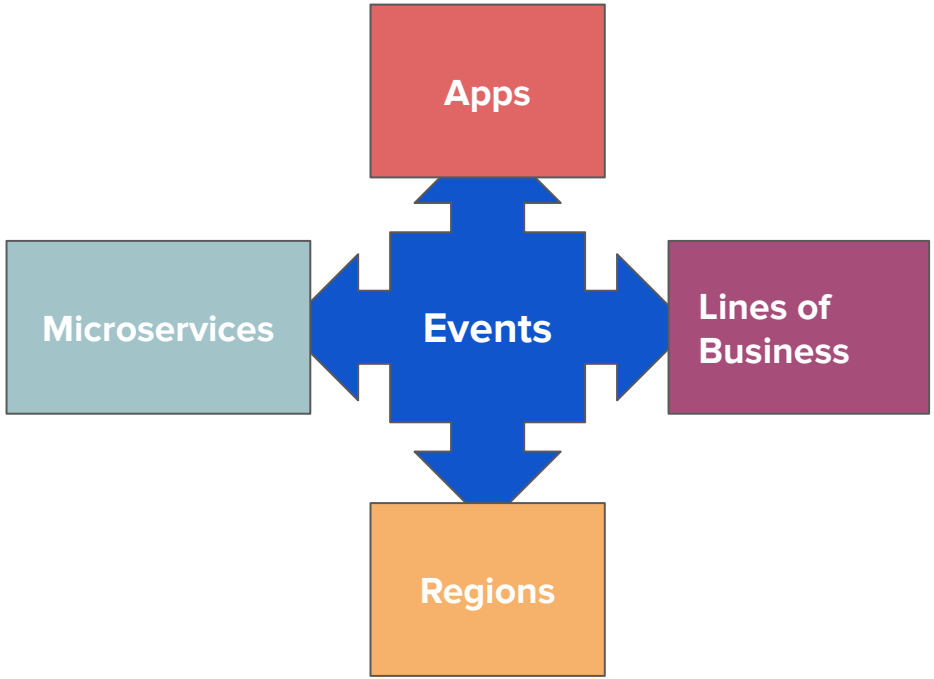
Events are ephemeral, intended to make systems work, while systems own their own systems of record

Red Hat

A Change in Perspective

Event-centric

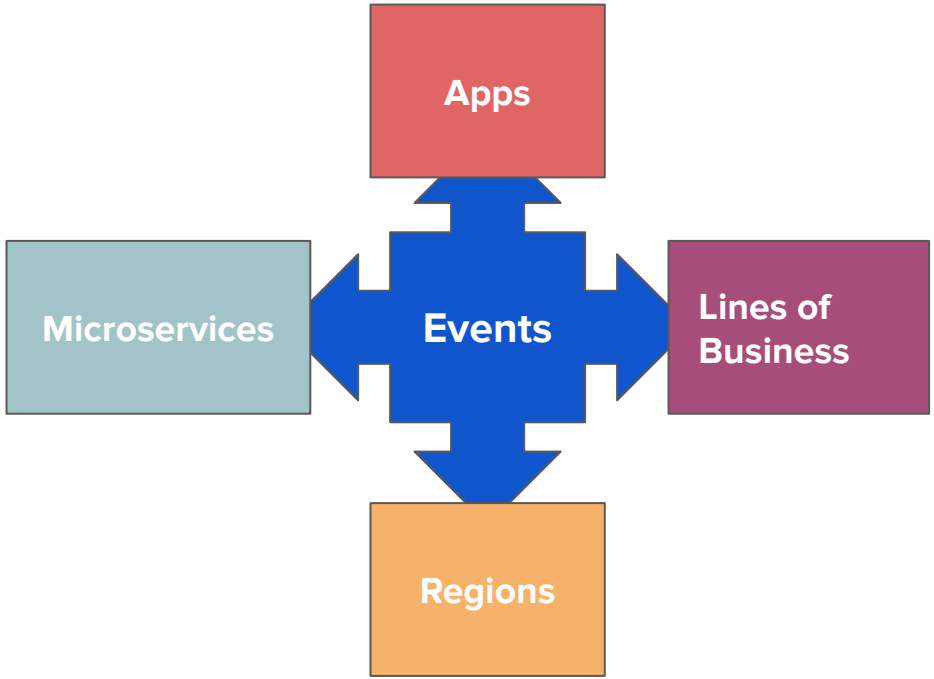
Events are long-lived or permanent; designed to serve as a first-class enterprise information store



A Change in Perspective

Advantages of this approach:

- Services can be simpler & stateless
- Communication patterns are clearer and easier to follow
- Data silos can be decreased while keeping ownership clear



So…in summary…

Architecture Pain Points	Architecture Solution
Tight Coupling	
Cascading Failures	
Call Chain Latency	
Cloud Latency	
Sizing/Scaling	

So…in summary…

Architecture Pain Points	Architecture Solution
Tight Coupling	
Cascading Failures	
Call Chain Latency	
Cloud Latency	
Sizing/Scaling	Serverless Deployment

So…in summary…

Architecture Pain Points	Architecture Solution
Tight Coupling	Event-Driven Architecture
Cascading Failures	
Call Chain Latency	
Cloud Latency	
Sizing/Scaling	Serverless Deployment

Red Hat

So…in summary…

Architecture Pain Points	Architecture Solution
Tight Coupling	Event-Driven Architecture
Cascading Failures	Event-Driven Architecture
Call Chain Latency	
Cloud Latency	
Sizing/Scaling	Serverless Deployment

So…in summary…

Architecture Pain Points	Architecture Solution
Tight Coupling	Event-Driven Architecture
Cascading Failures	Event-Driven Architecture
Call Chain Latency	Event-Driven Architecture
Cloud Latency	
Sizing/Scaling	Serverless Deployment

So…in summary…

Architecture Pain Points	Architecture Solution
Tight Coupling	Event-Driven Architecture
Cascading Failures	Event-Driven Architecture
Call Chain Latency	Event-Driven Architecture
Cloud Latency	Event-Driven Architecture
Sizing/Scaling	Serverless Deployment

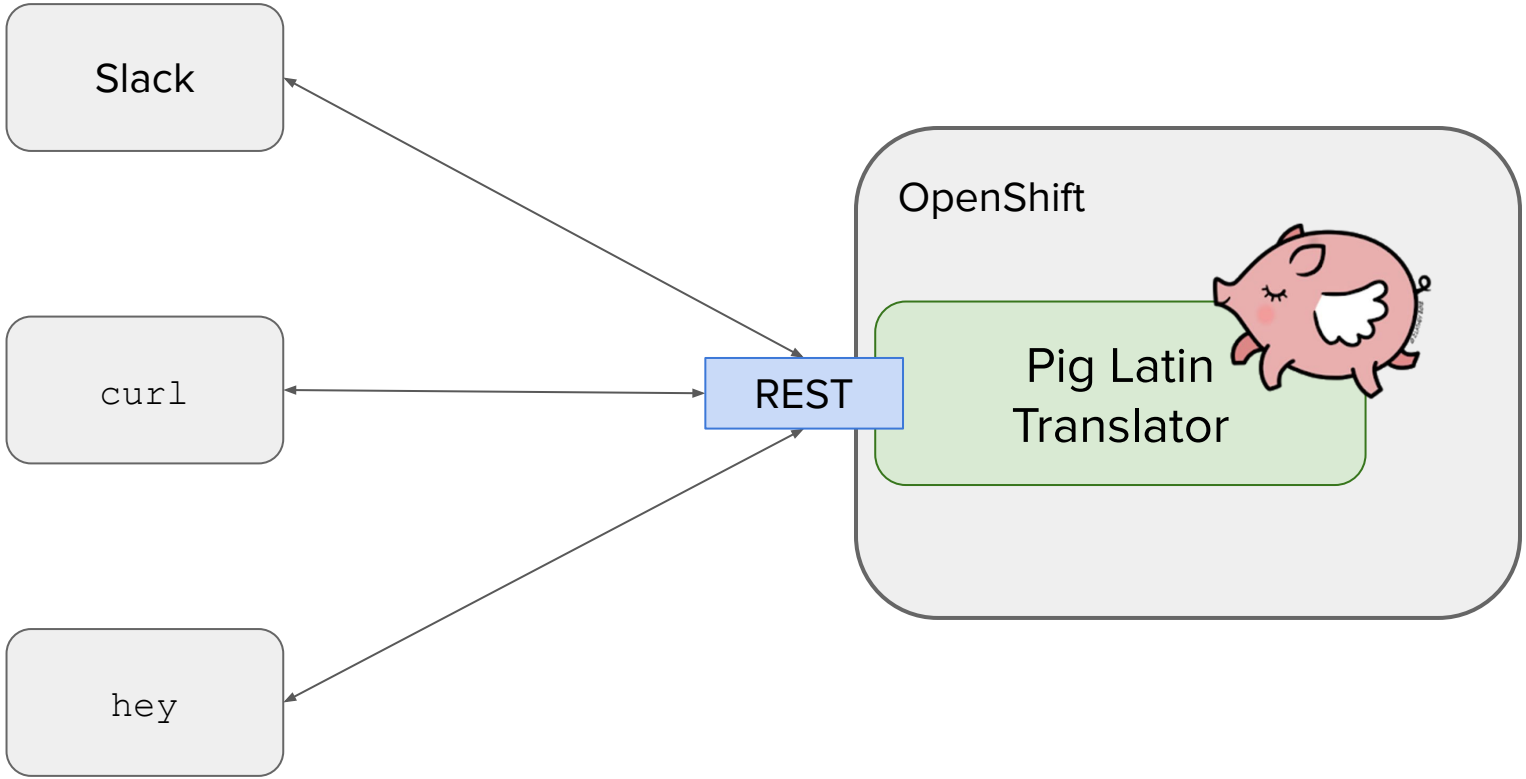
Red Hat

The Lab - Application Architecture

Red Hat

The Lab Application

- We're going to build an application that translates English to Pig Latin!
- That application will run in a container, both serverless-ly and *not* serverless-ly.
- It will be invoked via a Slack command


Slack
curl
hey
REST
OpenShift
Pig Latin
Translator

The Tools We're Using

- Slack
- Quarkus
- OpenShift
- OpenShift Serverless (Knative)

Let’s get you user ID’s!

- Go here: http://etherpad-labs.apps.cluster-ptd-80d5.ptd-80d5.example.opentlc.com/p/workshops
- Sign up for an unused user
- Use your user<#> and password openshift to log into OpenShift console
- https://console-openshift-console.apps.cluster-ptd-80d5.ptd-80d5.example.opentlc.com

Red Hat

Let’s Tour the Code!

Quarkus + Java, located here:

https://github.com/redhat-partner-tech/partner-tech-days-march2021/event-driven-serverless-openshift

Red Hat

Installing Knative

We only need to do this once, for the whole OpenShift cluster.

Basically we're flipping a switch that says, "allow applications to be deployed in a serverless way."

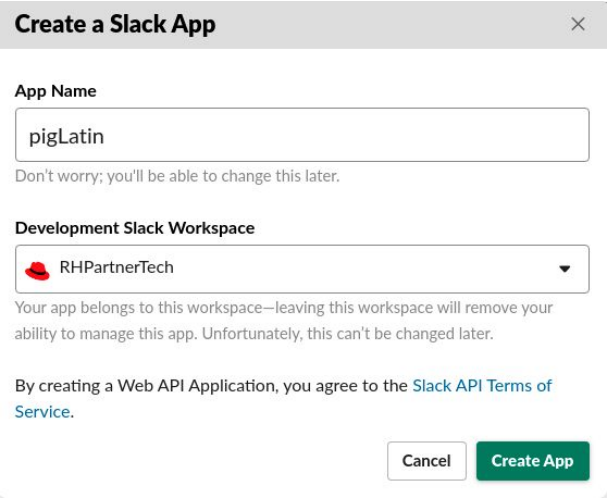
- Install the operator for all namespaces
- Make a project called **`knative-serving`**
- Create an instance of the **`knative-serving`** CRD in that project

NOTE: Knative has already been setup in the lab

Red Hat

Creating Slack App

- Sign up for slack and create a slack workspace (https://slack.com)
- Create a new App (https://api.slack.com/apps)



Red Hat

Creating Slack App

- Retrieve and save the slack app signing secret:

App Credentials

These credentials allow your app to access the Slack API. They are secret. Please don't share your app credentials with anyone, include them in public code repositories, or store them in insecure ways.

App ID

A01PC4KPR5M

Date of App Creation

March 1, 2021

Client ID

1750226992341.1794155807191

Client Secret

••••••••••

Show Regenerate

You'll need to send this secret along with your client ID when making your oauth.v2.access request.

Signing Secret

efa0dd536e1655caed1ef21828387d38

Show Regenerate

Slack signs the requests we send you using this secret. Confirm that each request comes from Slack by verifying its unique signature.

Red Hat

Deploy the Application

- Make a project - user<#>-piglatin
- Apply the Slack signing secret to the project
  - Workloads > Secret > Create > Key/Value Secret
  - Name: slack-signing-secret
  - Key: SLACK_SIGNING_SECRET
  - Value: <Signing Secret>

Deploy the Application

- Do a build using the git URL
  - Switch to Developer View
  - Select “+Add” menu
  - From Catalog > search for “openjdk” > OpenJDK Template (make sure Type Operator Backed is unchecked)

Red Hat

Application Name *

piglatin-app

The name for the application.

Java Version *

latest

The version of Java to use, e.g. 8, 11, latest. (Corresponds to the 'java' ImageStream tag.)

Git Repository URL *

https://github.com/redhat-partner-tech/partner-tech-days-march2021.git

Git source URI for application

Git Reference

main

Git branch/tag reference

Context Directory

event-driven-serverless-openshift/part1

Path within Git project to build; empty for root project directory.

Red Hat

Deploy the Application

- The template will build and deploy the application for you, creating and connecting the necessary components.
- Retrieve the route url for the newly deployed piglatin app

Setting up Slack

- Create a new slash command (https://api.slack.com/apps/<app id>)
- Select Slash Commands on the left menu
- Use the route URL for request URL and append "/events"

Command	/piglatin
Request URL	http://piglatin-app-user1-piglatin.ap...
Short Description	Translate to pig latin

Red Hat

Connect the Dots and Test it!

- Install the app to your slack workspace
- Test it in your slack workspace
  - Select pigLatin App in your slack workspace
  - Type “/piglatin Hello!” in the message box

Red Hat

Deploy the Application...again!

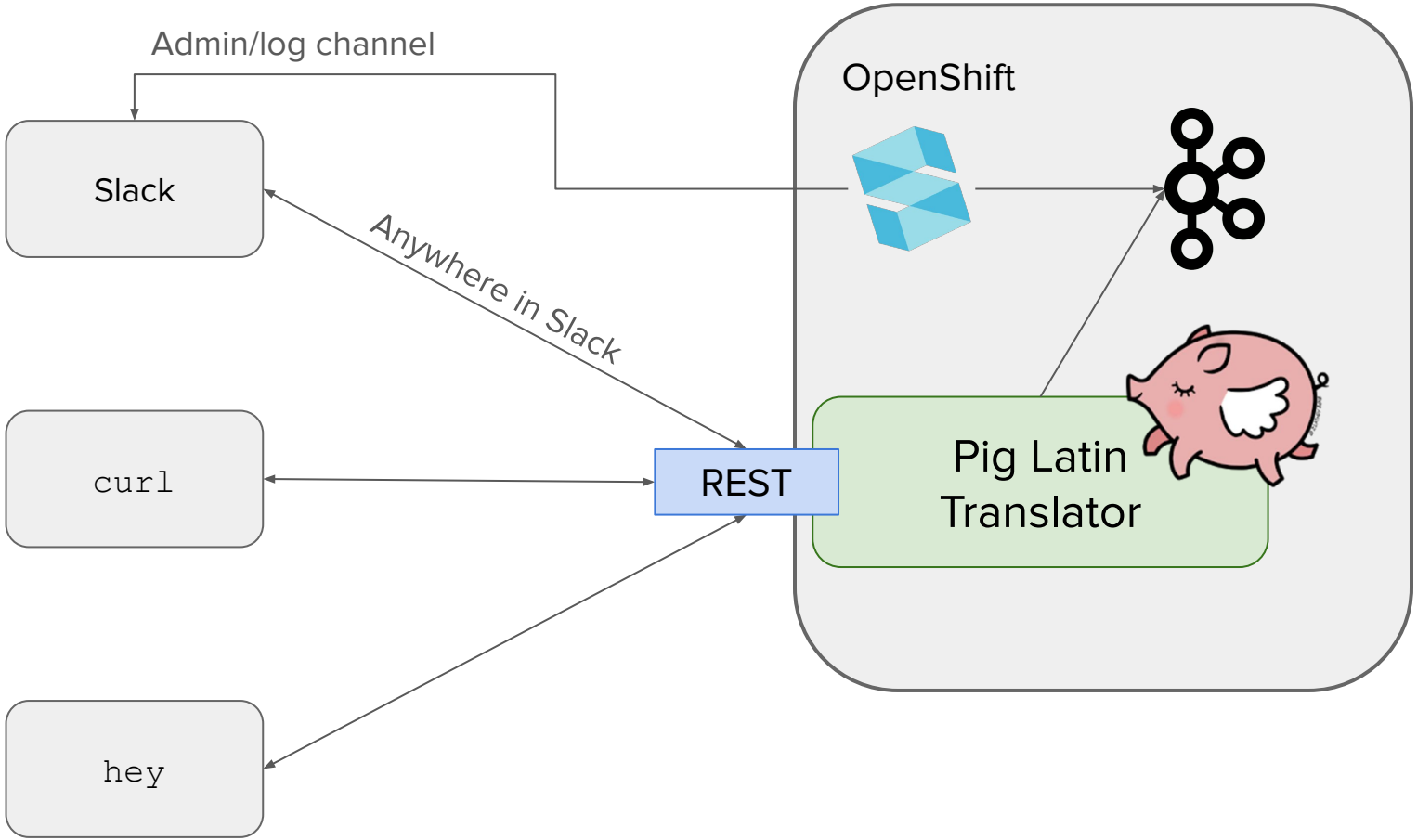
- Create a Serverless deployment of the application - using the same built image, tada!
- Grab the content of pl-serverless.yaml - don't forget to change the "userx-piglatin" to your project!
- Switch to Administrator View
- Go to Serverless > Services > Create Services and paste the content and complete the creation of the Service

Red Hat

Connect the Dots and Test it!

- Update Request URL of Slack app to use serverless version: https://api.slack.com/apps/<app id>/slash-commands
- /piglatin GO
- Watch it scale live!
- Note: If you get a timeout on the first try, repeat the operation again.

Red Hat


Admin/log channel
OpenShift
Slack
Anywhere in Slack
curl
REST
Pig Latin
Translator
hey

New Tools We're Using

- AMQ Streams (Kafka)
- Fuse Online (Syndesis)

Installing AMQ Streams (Kafka)

Everyone will use ONE Kafka cluster, with one topic. This is for simplicity's sake, not a requirement!

- Install the operator for all namespaces
- Make a project called **`amq-streams`**
- Make a Kafka cluster called... **`cluster`**
- Make a topic called **`slack`**

NOTE: AMQ Stream has already been setup in the lab

Red Hat

Installing Fuse Online (Syndesis)

Everyone will use ONE instance of Fuse Online.

- Make a project called **`fuse-online`**
- Install the operator in that project
- Make a “Fuse Online” object

NOTE: Fuse Online has already been setup in the lab

Enable Slack Webhook

- Enable Slack App Webhook (https://api.slack.com/apps/<app id>)
- Select Incoming Webhooks on the left menu
- Turn on and select Add New Webhook to Workspace at the bottom and select an “admin” slack channel for the webhook
- Retrieve the webhook URL

Webhook URL	Channel
https://hooks.slack.com/service Copy	#event-driven-architecture

Red Hat

Configuring Fuse Online

- Define the connection to the Kafka cluster
  - `cluster-kafka-bootstrap.amq-streams.svc:9092`

  NOTE: Kafka connection has already been setup in the lab

- Define and start up the integration between the two – this will be a JSON instance, defined very simply as: **`{"text":"text"}`**
- Access Fuse Online console

  (https://syndesis-fuse-online.apps.cluster-ptd-80d5.ptd-80d5.example.opentlc.com)

Red Hat

Configuring Fuse Online

- Select Connections and click Create Connection
- Define an HTTPS connection in Fuse Online:

HTTPS

All fields are required.

Base URL ⓘ *

https://hooks.slack.com/services/T01N26NV6A1/

Validate

- The base URL is the first 3 parts of your slack webhook URL

Red Hat

Configuring Fuse Online

- Define an HTTPS POST integration:

Invoke URL - properties

Http Method ⓘ POST

URL Path ⓘ /B01Q4FDRPV5/fMi3kuXbBm08kDfBuLjZ4xlx

< Choose Action | Next

- The post URL path should be the last 2 parts of your slack webhook URL

Red Hat

Configuring Fuse Online

- For the data type of the HTTPS integration, specify:

Deploy the Application

- Build the application from the Git URL
- This will build the application with Kafka enabled connectivity

Application Name *

piglatin-app-2

The name for the application.

Java Version *

latest

The version of Java to use, e.g. 8, 11, latest. (Corresponds to the 'java' ImageStream tag.)

Git Repository URL *

https://github.com/redhat-partner-tech/partner-tech-days-march2021.git

Git source URI for application

Git Reference

main

Git branch/tag reference

Context Directory

event-driven-serverless-openshift/part2

Path within Git project to build; empty for root project directory.

Red Hat

Deploy the Application

- Create a Serverless deployment of the application
  - Add to cluster using the pl-serverless.yaml file as a model - don't forget to change the "userx-piglatin" to your project!

Red Hat

Connect the Dots and Test it!

- Updates in Slack App:

  https://api.slack.com/apps/<app-id>/slash-commands
- /piglatin GO
- Check the admin Slack channel to see the message log

Red Hat

Thank you

Red Hat is the world's leading provider of enterprise open source software solutions. Award-winning support, training, and consulting services make Red Hat a trusted adviser to the Fortune 500.

linkedin.com/company/red-hat

youtube.com/user/RedHatVideos

facebook.com/redhatinc

twitter.com/RedHat

Red Hat