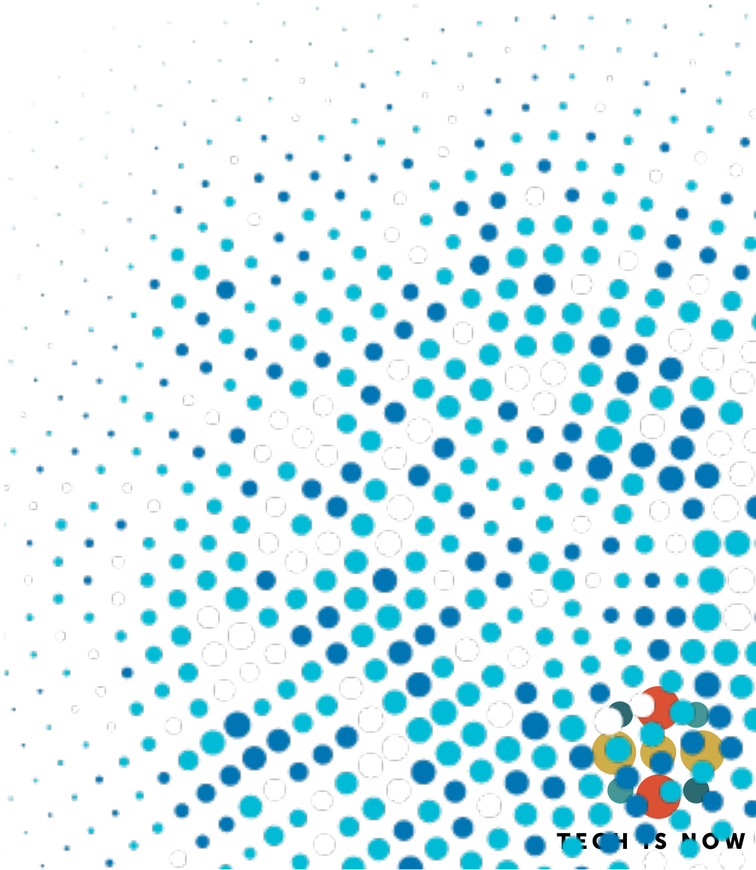
Desarrollo de Microservices Cloud Native

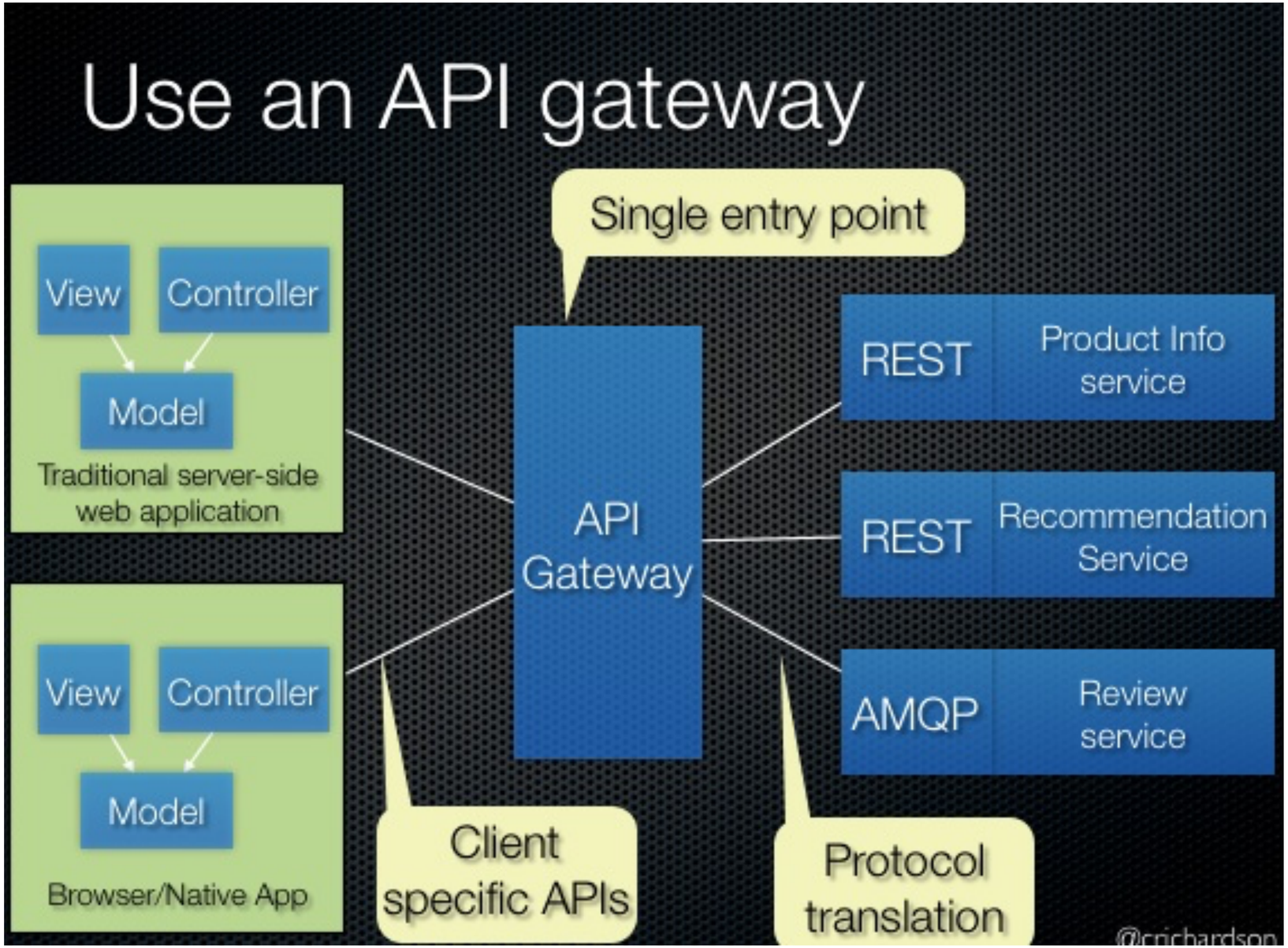


TECH IS NOW

API Gateway

API gateway

- ¿Cómo hacen los clientes de los Microservicios para acceder a servicios individuales?
- La granularidad de APIs que son proveídas por Microservices a menudo es muy diferente a lo que un cliente necesita.
- El número de instancias de los servicios y sus ubicaciones cambian dinámicamente.
- El particionado en servicios puede cambiar en el tiempo y debe ser ocultado de los clientes.

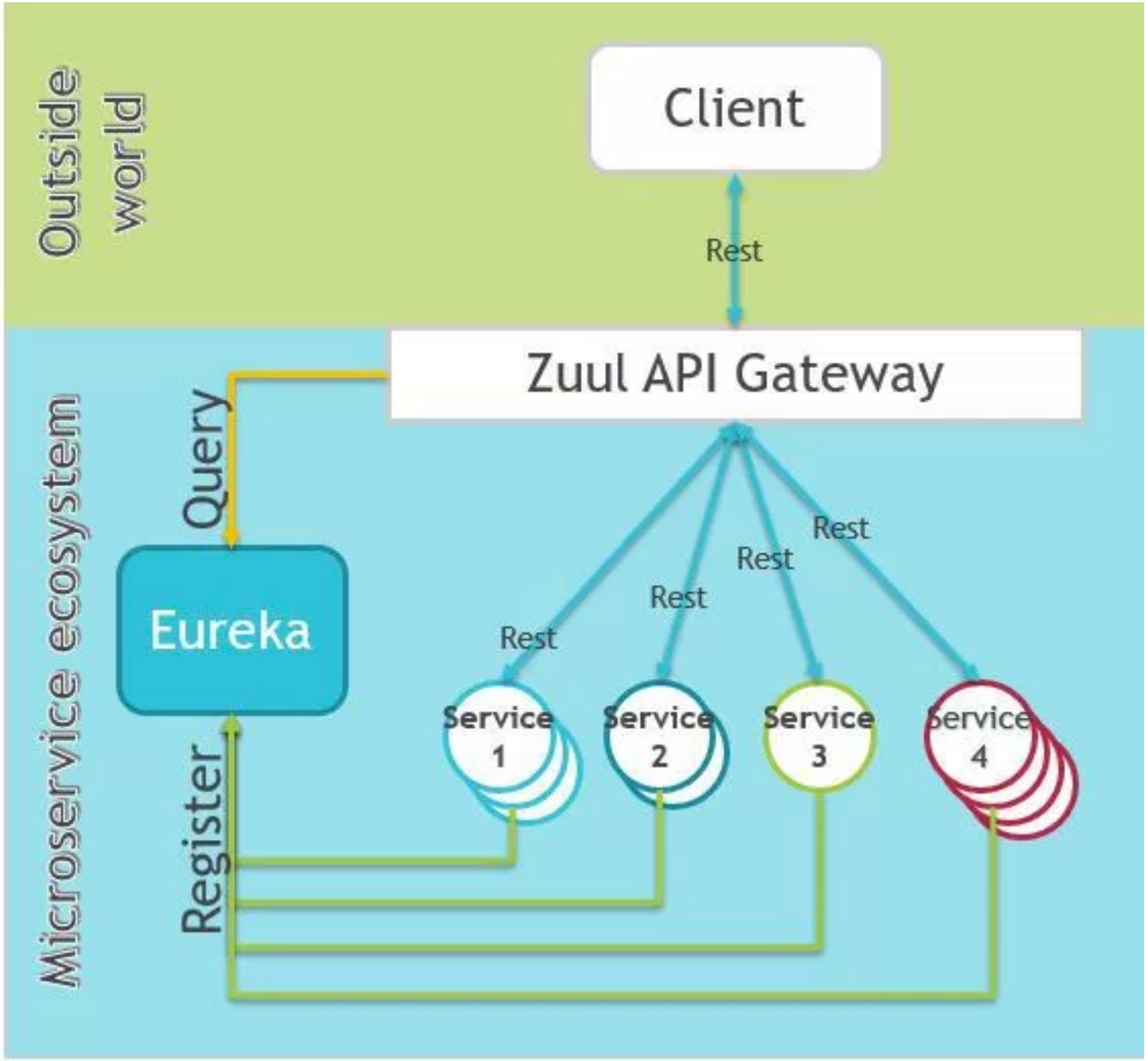

Use an API gateway
Single entry point
View
Controller
Model
Traditional server-side web application
View
Controller
Model
Browser/Native App
API Gateway
REST
Product Info service
REST
Recommendation Service
AMQP
Review service
Client specific APIs
Protocol translation
@crichardson

Netflix Zuul

TECH IS NOW

Zuul

- Una parte importante para una arquitectura de microservicios es el enrutado de las llamadas para llevar las peticiones de los usuarios a los microservicios correspondientes.
- Para ello existe Zuul, un enrutador y balanceador de carga en la parte del servidor para la JVM.
- Se integra con Eureka de tal manera que enrutará llamadas a urls con servicios Eureka, haciendo transparente a cuál de todas las instancias del mismo servicio está enrutando la llamada.


Outside world
Client
Rest
Zuul API Gateway
Microservice ecosystem
Query
Eureka
Register
Rest
Rest
Rest
Rest
Service 1
Service 2
Service 3
Service 4

TECH IS NOW

Features de Zuul

- Provee acceso unificado a diferentes Microservicios
- Oculta los detalles internos del ecosistema de Microservicios
- Balancea la carga entre las diferentes instancias
- Permite el acceso a los servicios
- Restringe el acceso a solamente los servicios internos
- Localiza los servicios de Eureka
- Se pueden implementar filtros para autenticación o bitacorado

clase05/01_zuul

TECH IS NOW

Feign (OpenFeign)

TECH IS NOW

Feign (OpenFeign)

- Feign es un cliente HTTP en Java inspirado por Retrofit, JAXRS-2.0, y WebSocket. El diseño principal de Feign es reducir la complejidad del binding de APIs REST.

Feign

- Feign procesa anotaciones de un request de un servicio REST a manera de plantilla.
- Antes de enviar la petición al servidor, los parámetros y argumentos son aplicados a esas plantillas.
- La limitación principal de Feign es que está por ahora con soporte de APIs basados en texto (JSON, XML).

```java
interface GitHub {
  @RequestLine("GET /repos/{owner}/{repo}/contributors")
  List<Contributor> contributors(@Param("owner") String owner, @Param("repo") String repo);
}

static class Contributor {
  String login;
  int contributions;
}

public static void main(String... args) {
  GitHub github = Feign.builder()
                       .decoder(new GsonDecoder())
                       .target(GitHub.class, "https://api.github.com");

  // Fetch and print a list of the contributors to this library.
  List<Contributor> contributors = github.contributors("netflix", "feign");
  for (Contributor contributor : contributors) {
    System.out.println(contributor.login + " (" + contributor.contributions + ")");
  }
}
```

Feign y Hystrix

- Crear clientes REST es muy sencillo.
- Solo hay que crear una interface y añadir algunas anotaciones. Feing creará la implementación en tiempo de ejecución, encontrará el endpoint desde el registro de Eureka.
- Feign proporciona soporte a Hystrix para hacer que los clientes de los servicios sean resilientes.

Feign (OpenFeign)

Feign (OpenFeign)

- Feign es un cliente HTTP en Java inspirado por Retrofit, JAXRS-2.0, y WebSocket. El diseño principal de Feign es reducir la complejidad del binding de APIs REST.

Feign

- Feign procesa anotaciones de un request de un servicio REST a manera de plantilla.

- Antes de enviar la petición al servidor, los parámetros y argumentos son aplicados a esas plantillas.

- La limitación principal de Feign es que esta por ahora con soporte de APIs basados en texto (JSON, XML).

```
interface GitHub {
  @RequestLine("GET /repos/{owner}/{repo}/contributors")
  List<Contributor> contributors(@Param("owner") String owner, @Param("repo") String repo);
}

static class Contributor {
  String login;
  int contributions;
}

public static void main(String... args) {
  GitHub github = Feign.builder()
                       .decoder(new GsonDecoder())
                       .target(GitHub.class, "https://api.github.com");

  // Fetch and print a list of the contributors to this library.
  List<Contributor> contributors = github.contributors("netflix", "feign");
  for (Contributor contributor : contributors) {
    System.out.println(contributor.login + " (" + contributor.contributions + ")");
  }
}
```

Feign y Hystrix

- Crear clientes REST es muy sencillo.

- Solo hay que crear una interface y añadir algunas anotaciones. Feing creará la implementación en tiempo de ejecución, encontrará el endpoint desde el registro de Eureka.

- Feign proporciona soporte a Hystrix para hacer que los clientes de los servicios sean resilientes.

Estrategia de pruebas

Pruebas

- Hacer la mayor cantidad de pruebas de unitarias
  - Mocks/Stubs
  - No es unitaria si se levanta el Application Context de Spring
- Probar de forma integral lo más pronto posible
- Usar “feature branches” en Git
  - Si se combina con el Pipeline multibranch de Jenkins queda mucho mejor.

Spring Rest Docs

TECH IS NOW

Spring Security

TECH IS NOW

Seguridad

- Agregar starter de Spring Security
  - compile “org.springframework.boot:spring-boot-starter-security"

```
@Configuration
@EnableWebSecurity
class WebSecurityConfig extends WebSecurityConfigurerAdapter {

  @Override
  protected void configure(HttpSecurity http) throws Exception {
    http.authorizeRequests().antMatchers("/static/**").permitAll()
      .anyRequest().fullyAuthenticated()
      .and().httpBasic()
      .and().csrf().disable()
  }

  @Autowired
  void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
    UserDetailsManagerConfigurer authentication =
      auth.inMemoryAuthentication()

    authentication.withUser(“username”).password(“password").roles("USER")
  }
}
```

Deployment

TECH IS NOW

Servidor tradicional

- El procedimiento aplica por cada aplicación, archivo jar.
- Preparar el jar que sea autoejecutable

```
bootJar {
  launchScript()
}
```

- Se puede agregar como servicio del sistema si se hace un enlace simbólico en el equipo
  - ***sudo ln -s /opt/app/app.jar /etc/init.d/app***
- Se debe dar permisos de ejecución al JAR

Configuración

- Escribir un archivo con extension *conf* en el mismo directorio del jar
  - app.jar —> app.conf
  - Dicho archivo contiene los parámetros de ejecución de la JVM

```
JAVA_OPTS="-Djava.security.egd=file:/dev/./urandom -Xms256m -
Xmx256m"
RUN_ARGS=--spring.rabbitmq.password=3ch8GuTTZ9N4QgBQWfZV
```

```
JAVA_OPTS="-Djava.security.egd=file:/dev/./urandom -Xms256m -Xmx256m"
RUN_ARGS=--spring.profiles.active=dev
```

Deployment con Docker Compose/Swarm

- Se recomienda crear un archivo de deployment (docker-compose.yml)
- Para tener el servicio arriba usar el atributo:
  - restart: always
- Aplicar la nueva configuración con algo como:

```
docker-compose -f docker-compose.yml -f
production.yml up -d
```

Opciones de Deployment

- Kubernetes
  - http://kubernetes.io/

- Amazon EC2 Container Service
  - https://aws.amazon.com/ecs/

Monitoreo

Spring Boot Admin
Revisión

TECH IS NOW

Spring Boot Admin

https://github.com/codecentric/spring-boot-admin

Administrar aplicaciones

- Configurar el servidor de Spring Boot Admin
  - compile('de.codecentric:spring-boot-admin-starter-server’)
- Configurar los clientes
  - compile('de.codecentric:spring-boot-admin-starter-client')
  - spring.boot.admin.url=http://{ip_server}:8080

Observability

Disclaimer

- En el contexto de Cloud Native, Observability no tiene nada que ver con el significado en teoría de control.
- https://en.wikipedia.org/wiki/Observability


Cindy Sridharan
@copyconstruct
OH - "Observability - because devs don't like to do "monitoring" we need to package it in new nomenclature to make it palatable and trendy."
4:41 PM - Jul 28, 2017 · San Francisco, CA
10
10
37


Cyfe
IT
Mar 11, 2014 - Apr 9, 2014
SERVER 1
Up
Last down 62 days ago
SERVER 2
Down
Last up 27 days ago
SERVER 3
Up
Last down 36 days ago
SERVER LOAD
Server 1 (1%)
Server 2 (0%)
Server 3 (1%)
Server 4 (15%)
Server 5 (82%)
SERVER UPTIME
0.372s ▼2%
RESPONSE TIME
99.99%
UPTIME
00:06:00 ▼25%
DOWNTIME
Mar 12 Mar 14 Mar 16 Mar 18 Mar 20 Mar 22 Mar 24 Mar 26 Mar 28 Mar 30 Apr 1 Apr 3 Apr 5 Apr 7
USERS BY SERVER
SERVER
USERS
1. Optimum 100,132
2. Crystal 52,035
3. Carter 10,342
4. Mona Lisa 5,413
5. Gator 12 100
SITE SPEED
SPEED TYPE
SECONDS
1. Page Load Time 2.89 ▲17%
2. Redirection Time 0.08 ▲700%
3. Domain Lookup Time 0.02 ▼75%
4. Server Connection Time 0.02
DARK

Monitoreo != Observabilidad

Métricas

- Mediciones en 3 niveles
  1. Red
  2. Maquina
  3. Aplicación
- Las métricas aplicativas son usualmente las más difíciles y las más importantes de las 3.

Monitoreo

- Recopilación de las métricas en rangos de tiempo.
- Almacenan en algún medio.
- Generan los dashboards.
- Es complejo hacer correlación de las métricas.
- https://codeascraft.com/2011/02/15/measure-anything-measure-everything/
- **El monitoreo es para alertas basadas en síntomas**

Monitoreo debe contestar:

- ¿qué está roto y por qué?
- El "qué está roto" indica el síntoma; el "por qué" indica una causa (posiblemente intermedia).
- "Lo que" VS “El por qué" es una de las distinciones más importantes al escribir un buen monitoreo con señal máxima y ruido mínimo.

TECH IS NOW

Retos del monitoreo

- La construcción de sistemas "monitoreables" requiere poder comprender el dominio de fallas de los componentes críticos del sistema de forma proactiva.
- Y esa es una tarea difícil. Especialmente para sistemas complejos.
- Más aún para sistemas simples que interactúan de manera compleja entre sí.

Observability

- Monitoreo
- Alerting/visualizaciones
- Infraestructura de trazabilidad de sistemas distribuidos
- Agregación de bitácoras / análisis

Observability es un superconjunto de Monitoreo

Observability VS Monitoreo

- Los objetivos de monitoreo y observabilidad son diferentes.
- La "capacidad de observación" no es un sustituto del "monitoreo" ni hace obvia la necesidad de “monitoreo”.
- Son complementarios.

Objetivo de Observability

- Proporcionar información muy detallada sobre el comportamiento de los sistemas junto con un contexto enriquecido, perfecto para la depuración.

Debugging

- Proceso iterativo, que implica la introspección iterativa de las diversas observaciones y hechos informados por el sistema
- Haciendo las deducciones correctas y probando si la teoría es correcta.
- La evidencia no se puede extraer de la nada ni se puede extrapolar a partir de agregados, promedios, percentiles, patrones históricos o cualquier otra forma de datos recolectados principalmente con fines de monitoreo.
- La evidencia debe ser informada por los sistemas en forma de hechos y observaciones altamente precisos y contextuales, que luego pueden usarse mientras se depura para teorizar sobre por qué algo podría no funcionar como se esperaba.

Observability VS fallas

- A diferencia del monitoreo que se conoce como *centrado en los fallos*, "Observability" no necesariamente tiene que estar estrechamente relacionado con una interrupción o una queja del usuario.
- Se puede utilizar como una forma de comprender mejor el desempeño y el comportamiento del sistema, incluso durante lo que se puede percibir como el funcionamiento "normal" de un sistema.

Herramientas


ZIPKIN


Grafana

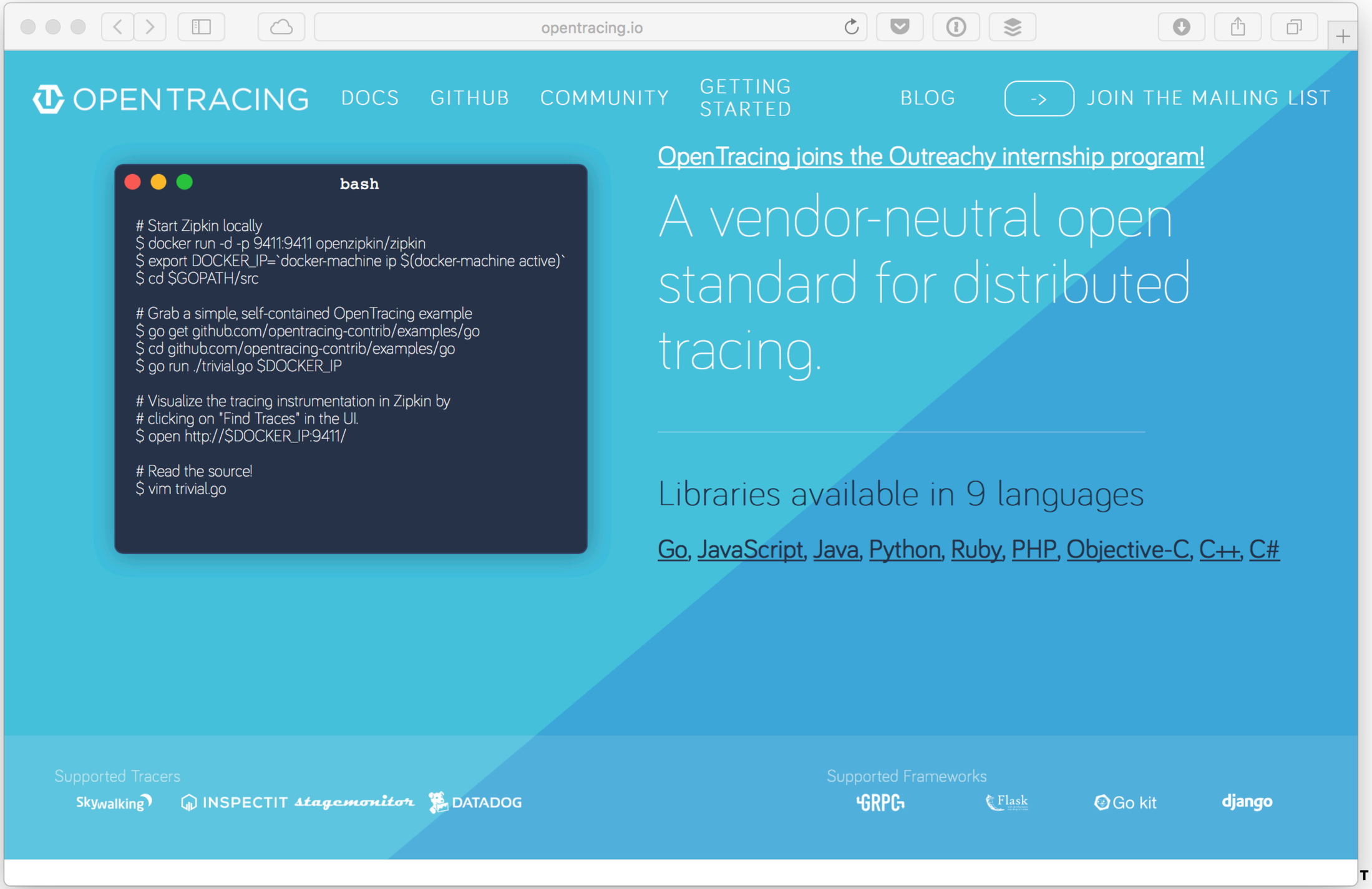

influxdb


JAEGER


fluentd


OPENTRACING

Prometheus


opentracing.io
OPENTRACING
DOCS
GITHUB
COMMUNITY
GETTING STARTED
BLOG
->
JOIN THE MAILING LIST
bash
# Start Zipkin locally
$ docker run -d -p 9411:9411 openzipkin/zipkin
$ export DOCKER_IP=`docker-machine ip $(docker-machine active)`
$ cd $GOPATH/src
# Grab a simple, self-contained OpenTracing example
$ go get github.com/opentracing-contrib/examples/go
$ cd github.com/opentracing-contrib/examples/go
$ go run ./trivial.go $DOCKER_IP
# Visualize the tracing instrumentation in Zipkin by
# clicking on "Find Traces" in the UI.
$ open http://$DOCKER_IP:9411/
# Read the source!
$ vim trivial.go
OpenTracing joins the Outreachy internship program!
A vendor-neutral open standard for distributed tracing.
Libraries available in 9 languages
Go, JavaScript, Java, Python, Ruby, PHP, Objective-C, C++, C#
Supported Tracers
Skywalking
INSPECTIT
stagemonitor
DATADOG
Supported Frameworks
GRPC
Flask
Go kit
django

TECH IS NOW

Monitoreo con Prometheus y Grafana

TECH IS NOW