สยามชำนาญกิจ
SIAM CHAMNAN KIT

Week 05

Topics

Application metric with Actuator
Working with Prometheus and Grafana
Domain-Driven Design
Event storming workshop
Q/A

Application metric with Actuator

https://docs.spring.io/spring-boot/docs/current/reference/html/actuator.html

Architecture

Add actuator and prometheus

spring initializr

Project
- (●) Maven Project
- ( ) Gradle Project

Language
- (●) Java
- ( ) Kotlin
- ( ) Groovy

Spring Boot
- ( ) 3.0.0 (SNAPSHOT)
- ( ) 3.0.0 (M1)
- ( ) 2.7.0 (SNAPSHOT)
- ( ) 2.7.0 (M2)
- ( ) 2.6.5 (SNAPSHOT)
- (●) 2.6.4
- ( ) 2.5.11 (SNAPSHOT)
- ( ) 2.5.10

Project Metadata

Group	com.example
Artifact	demo
Name	demo
Description	Demo project for Spring Boot
Package name	com.example.demo
Packaging	(●) Jar ( ) War
Java	( ) 17 (●) 11 ( ) 8

Dependencies ADD DEPENDENCIES... ⌘ + B

Spring Boot Actuator OPS
Supports built in (or custom) endpoints that let you monitor and manage your application - such as application health, metrics, sessions, etc.

Prometheus OBSERVABILITY
Expose Micrometer metrics in Prometheus format, an in-memory dimensional time series database with a simple built-in UI, a custom query language, and math operations.

https://start.spring.io/

Add actuator and prometheus

Config in file pox.xml

```xml
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>

<dependency>
    <groupId>io.micrometer</groupId>
    <artifactId>micrometer-registry-prometheus</artifactId>
    <scope>runtime</scope>
</dependency>
```

https://start.spring.io/

Enable prometheus

Enabled endpoint in application.properties

management.endpoints.web.exposure.include=*

Endpoint of prometheus

GET /actuator/prometheus

localhost:8080/actuator/prometheus

```
# HELP jvm_buffer_total_capacity_bytes An estimate of the total capacity of the buffers in this pool
# TYPE jvm_buffer_total_capacity_bytes gauge
jvm_buffer_total_capacity_bytes{id="mapped - 'non-volatile memory'",} 0.0
jvm_buffer_total_capacity_bytes{id="mapped",} 0.0
jvm_buffer_total_capacity_bytes{id="direct",} 24577.0
# HELP jdbc_connections_idle Number of established but idle connections.
# TYPE jdbc_connections_idle gauge
jdbc_connections_idle{name="dataSource",} 10.0
# HELP jvm_buffer_count_buffers An estimate of the number of buffers in the pool
# TYPE jvm_buffer_count_buffers gauge
jvm_buffer_count_buffers{id="mapped - 'non-volatile memory'",} 0.0
jvm_buffer_count_buffers{id="mapped",} 0.0
jvm_buffer_count_buffers{id="direct",} 4.0
# HELP http_server_requests_seconds
# TYPE http_server_requests_seconds summary
http_server_requests_seconds_count{exception="None",method="GET",outcome="SUCCESS",status="200",uri="
http_server_requests_seconds_sum{exception="None",method="GET",outcome="SUCCESS",status="200",uri="/a
# HELP http_server_requests_seconds_max
# TYPE http_server_requests_seconds_max gauge
http_server_requests_seconds_max{exception="None",method="GET",outcome="SUCCESS",status="200",uri="/a
# HELP jvm_gc_max_data_size_bytes Max size of long-lived heap memory pool
# TYPE jvm_gc_max_data_size_bytes gauge
jvm_gc_max_data_size_bytes 4.294967296E9
# HELP jdbc_connections_active Current number of active connections that have been allocated from the
# TYPE jdbc_connections_active gauge
jdbc_connections_active{name="dataSource",} 0.0
# HELP executor_active_threads The approximate number of threads that are actively executing tasks
```

Endpoint of prometheus

Default metrics

```
jvm_buffer_count_buffers{id="direct",} 7.0
# HELP http_server_requests_seconds
# TYPE http_server_requests_seconds summary
http_server_requests_seconds_count{exception="None",method="GET",outcome="SUCCESS",status="200",uri="/actuator/prometheus",} 2.0
http_server_requests_seconds_sum{exception="None",method="GET",outcome="SUCCESS",status="200",uri="/actuator/prometheus",} 0.125273652
http_server_requests_seconds_count{exception="None",method="GET",outcome="SUCCESS",status="200",uri="/demo",} 1.0
http_server_requests_seconds_sum{exception="None",method="GET",outcome="SUCCESS",status="200",uri="/demo",} 0.039481691
http_server_requests_seconds_count{exception="None",method="GET",outcome="SUCCESS",status="200",uri="/actuator",} 1.0
http_server_requests_seconds_sum{exception="None",method="GET",outcome="SUCCESS",status="200",uri="/actuator",} 0.222248964
# HELP http_server_requests_seconds_max
# TYPE http_server_requests_seconds_max gauge
http_server_requests_seconds_max{exception="None",method="GET",outcome="SUCCESS",status="200",uri="/actuator/prometheus",} 0.116286878
http_server_requests_seconds_max{exception="None",method="GET",outcome="SUCCESS",status="200",uri="/demo",} 0.039481691
http_server_requests_seconds_max{exception="None",method="GET",outcome="SUCCESS",status="200",uri="/actuator",} 0.222248964
# HELP jvm_gc_max_data_size_bytes Max size of long-lived heap memory pool
# TYPE jvm_gc_max_data_size_bytes gauge
```

Custom metric

Working with MicroMeter

```java
@RestController
public class LoginController {

    @Autowired
    private MeterRegistry meterRegistry;

    @GetMapping("/login/{status}")
    public String login(@PathVariable  String status) {
        meterRegistry.counter("login_count", "status", status).increment();
        return "TODO with " + status;
    }

}
```

Endpoint of prometheus

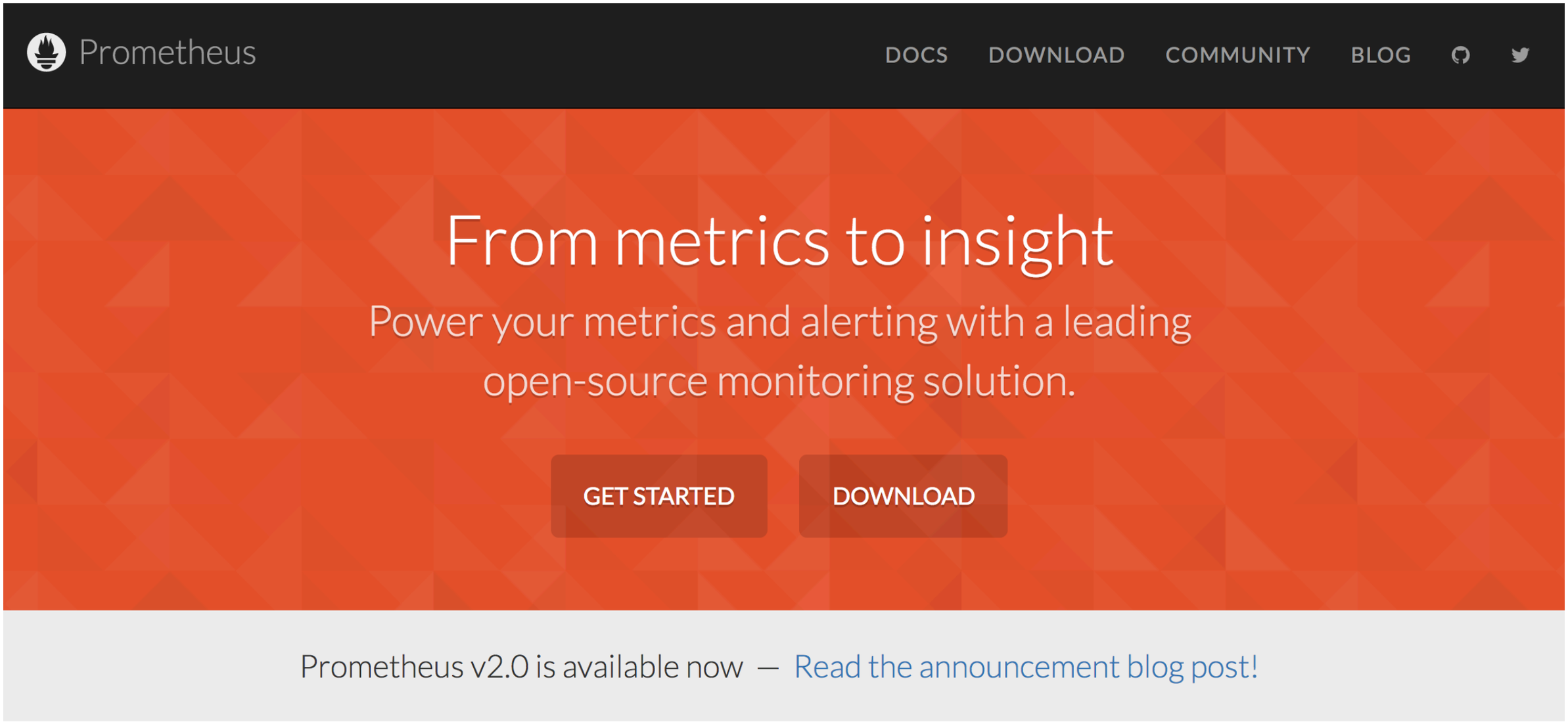
Metric name = login_count

```
# HELP login_count_total
# TYPE login_count_total counter
login_count_total{status="fail",} 2.0
login_count_total{status="success",} 3.0
```

Keep data in Prometheus

https://prometheus.io/

Prometheus



https://prometheus.io/

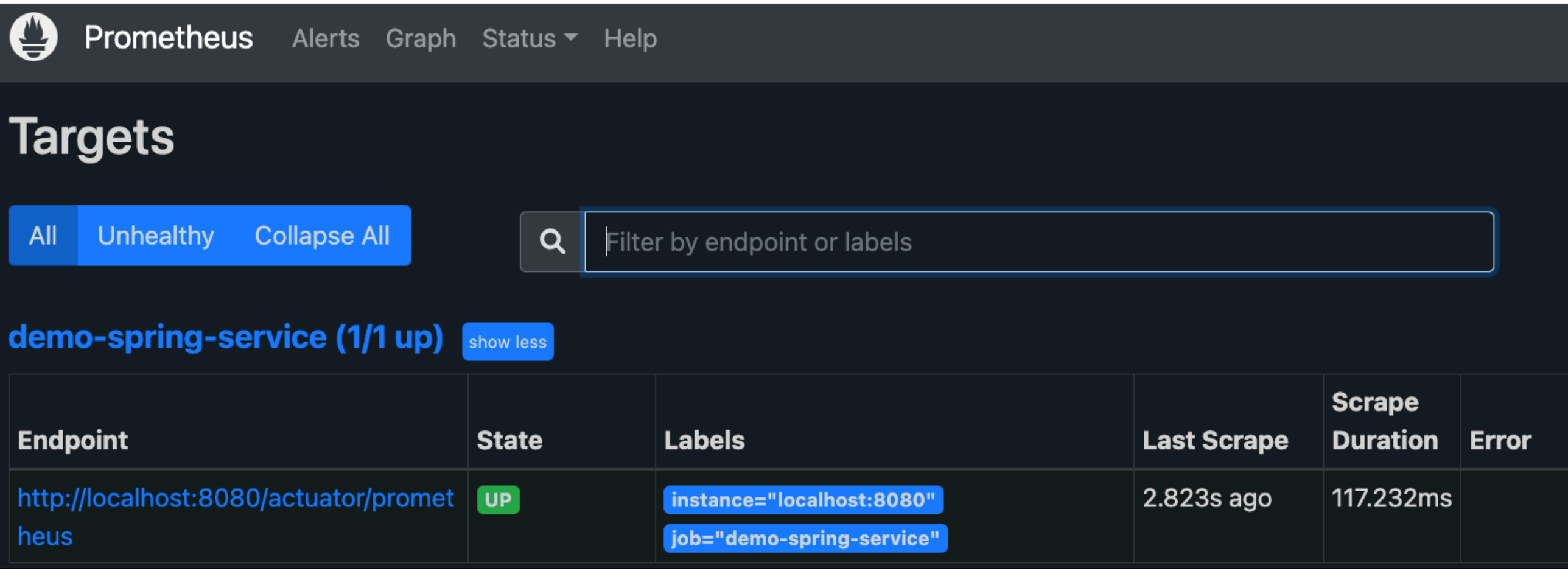
Config file prometheus.yml

```yaml
scrape_configs:
  # The job name is added as a label `job=<
  - job_name: "demo-spring-service"

    # metrics_path defaults to '/metrics'
    # scheme defaults to 'http'.
    metrics_path: '/actuator/prometheus'
    static_configs:
      - targets: ["localhost:8080"]
```

Check target

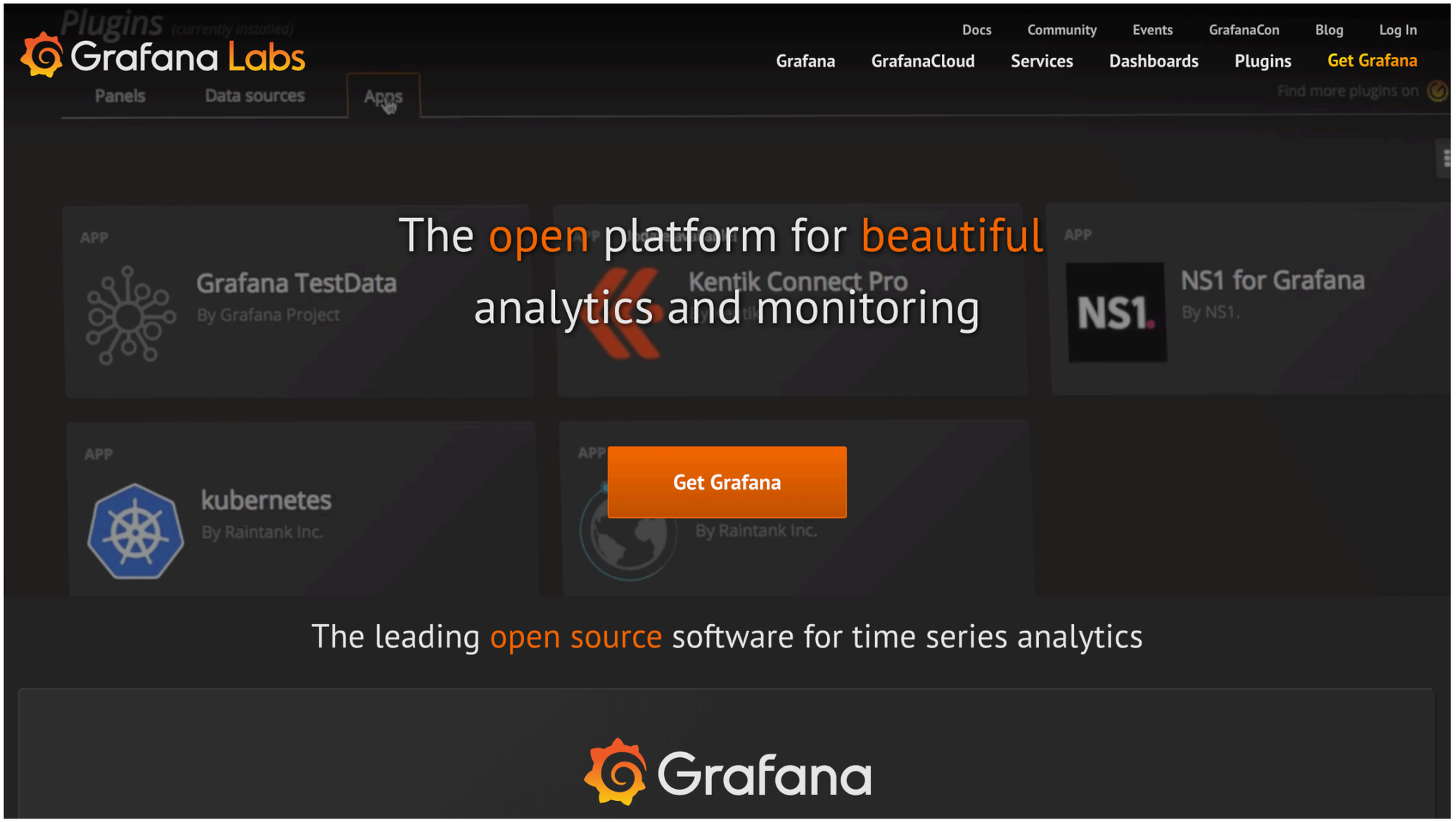
http://localhost:9090/targets



Show data in Grafana

https://grafana.com/

Grafana



https://grafana.com/

Grafana Dashboard

https://grafana.com/dashboards/4701

All dashboards » JVM (Micrometer)

JVM (Micrometer) by mweirauch

DASHBOARD

Dashboard for Micrometer instrumented applications (Java, Spring Boot)

Last updated: 21 days ago

Downloads: 74

Overview | Revisions

Get this dashboard:

4701

Copy ID to Clipboard

Download JSON

How do I import this dashboard?

A dashboard for Micrometer instrumented applications (Java, Spring Boot).

Features

- JVM memory
- Process memory (provided by micrometer-jvm-extras)
- CPU-Usage, Load, Threads, File Descriptors, Log Events
- JVM Memory Pools (Heap, Non-Heap)
- Garbage Collection

Dependencies:

GRAFANA 4.6.3

GRAPH

Domain-Driven Design

Domain-Driven Design

Problem space
Solution space

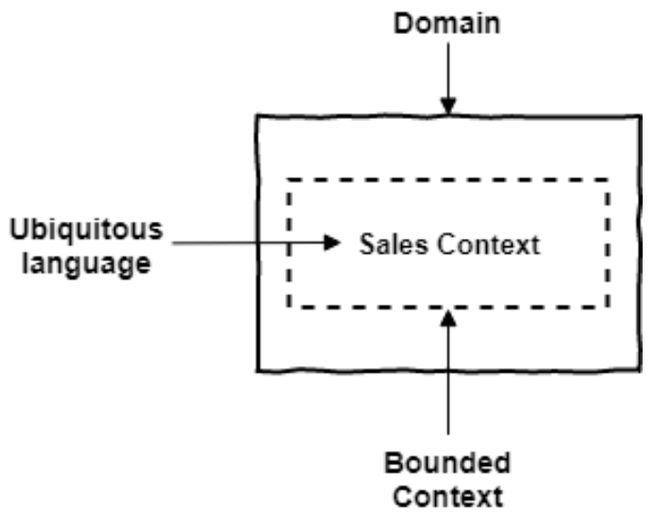
Problem space

Usages of customers

Words used by people and their meaning (Domain language)

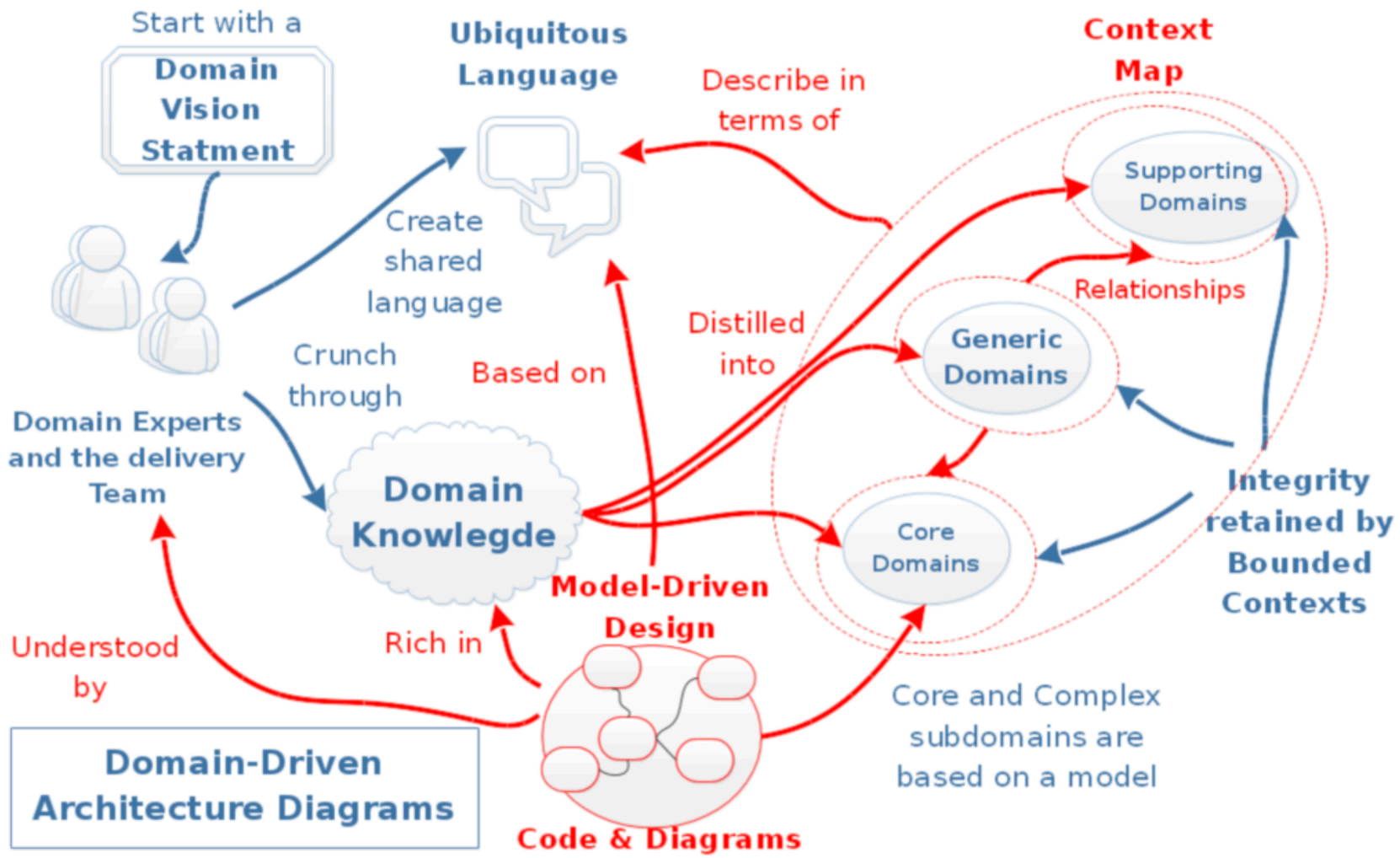
Requirements and **constraints** of business

People who operate business



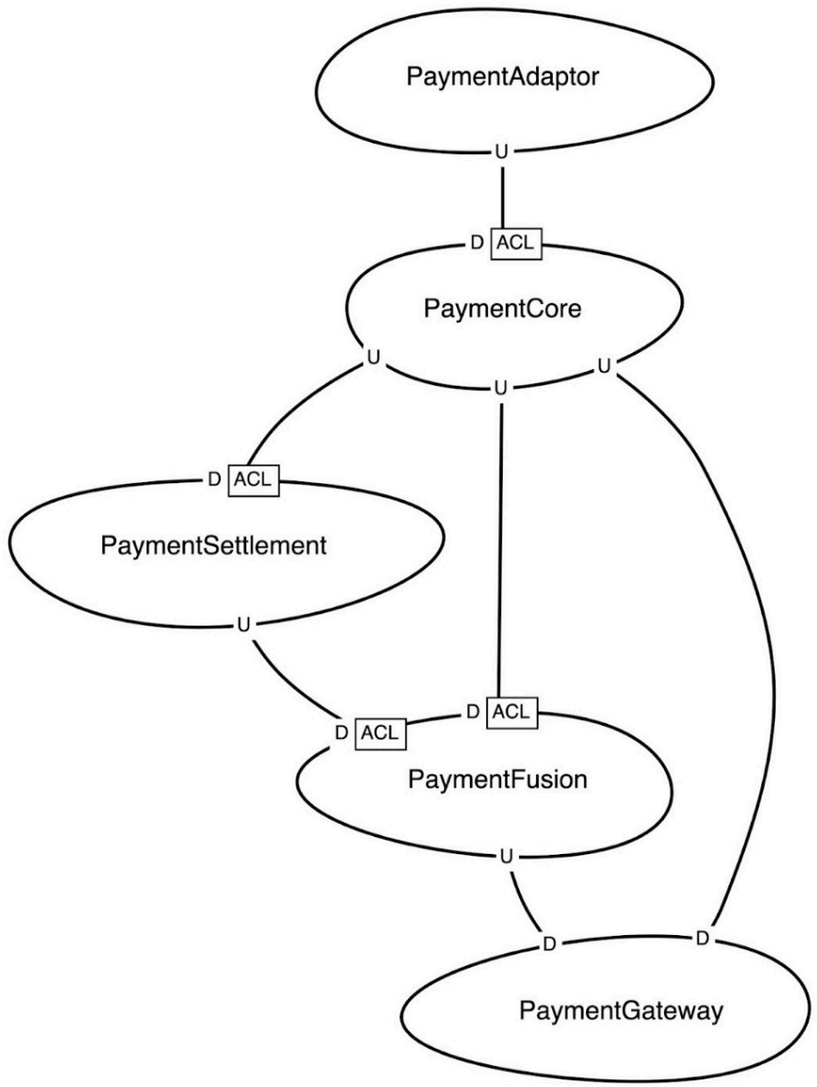
Problem space

Define structure of system
With ***strategic patterns***



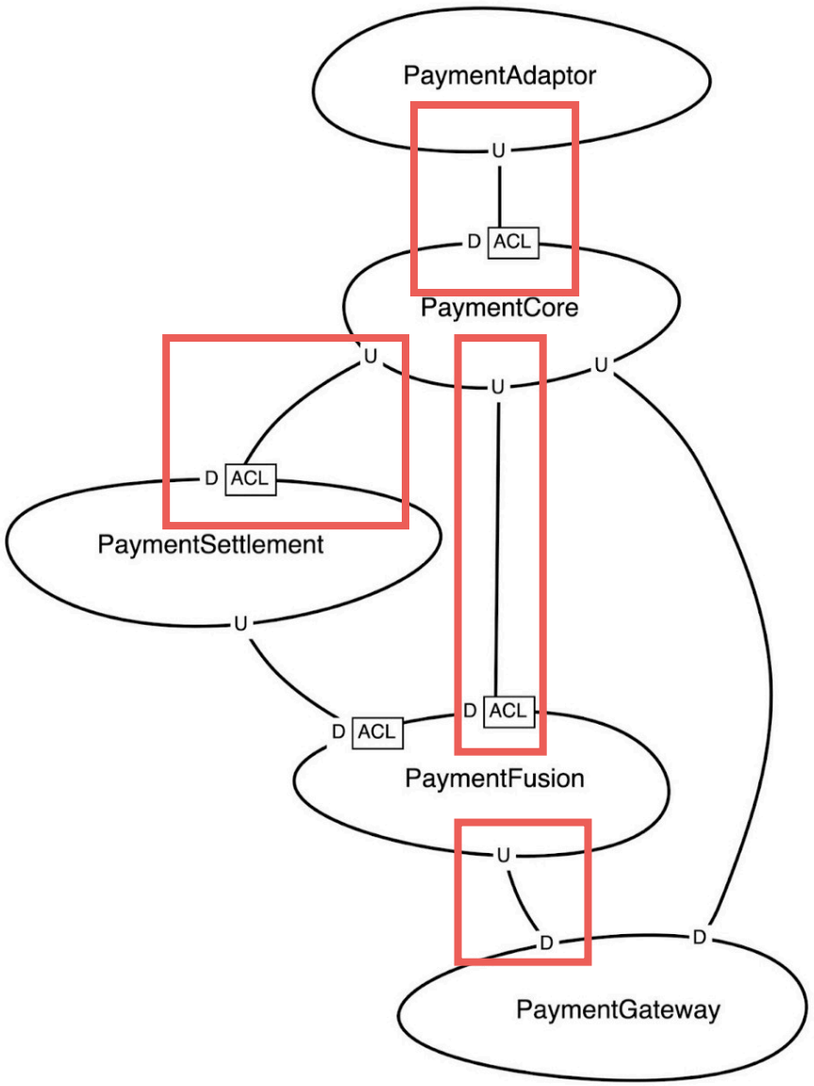
Solution space

Boundary Context (BC) and Context Map

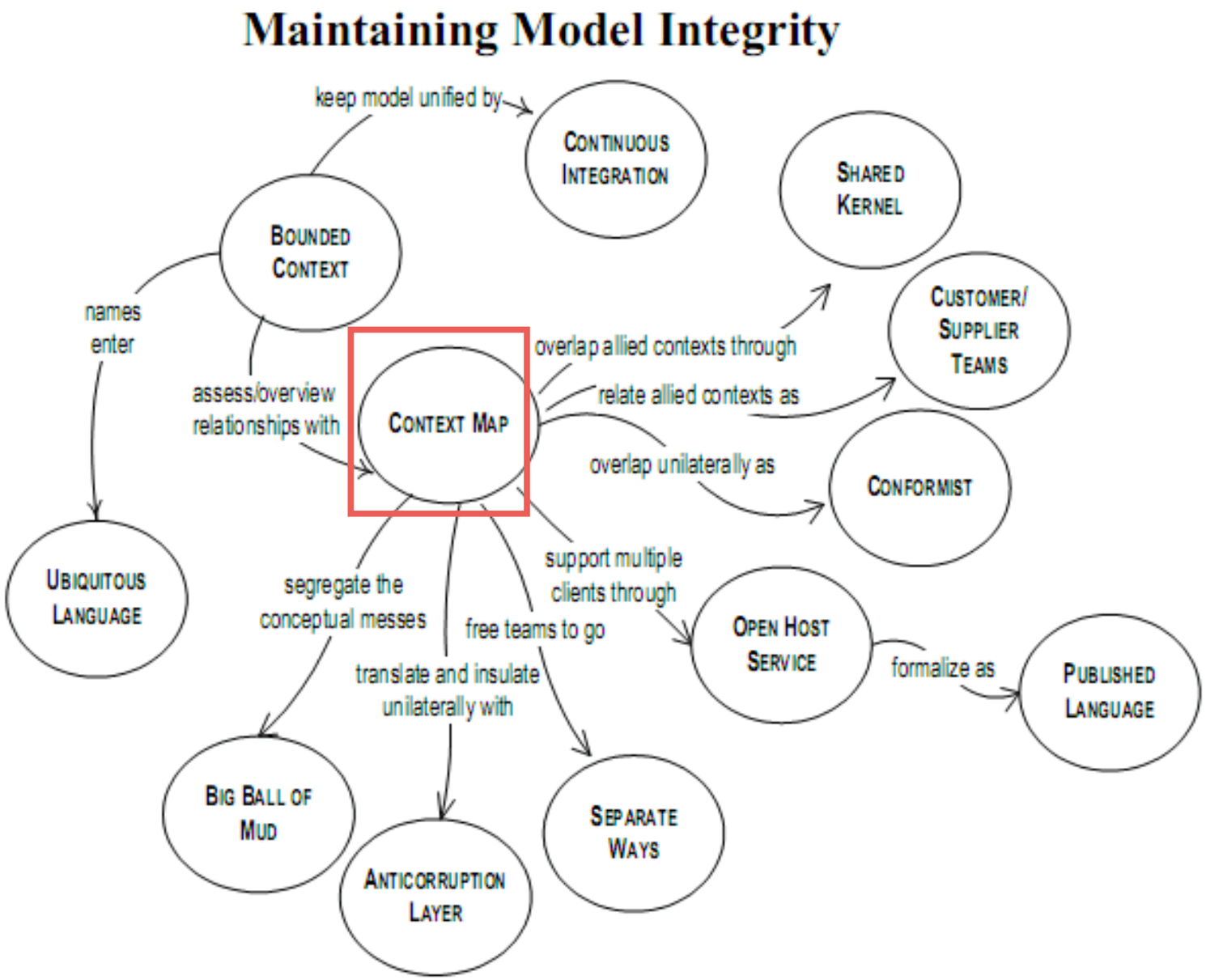


Context Map

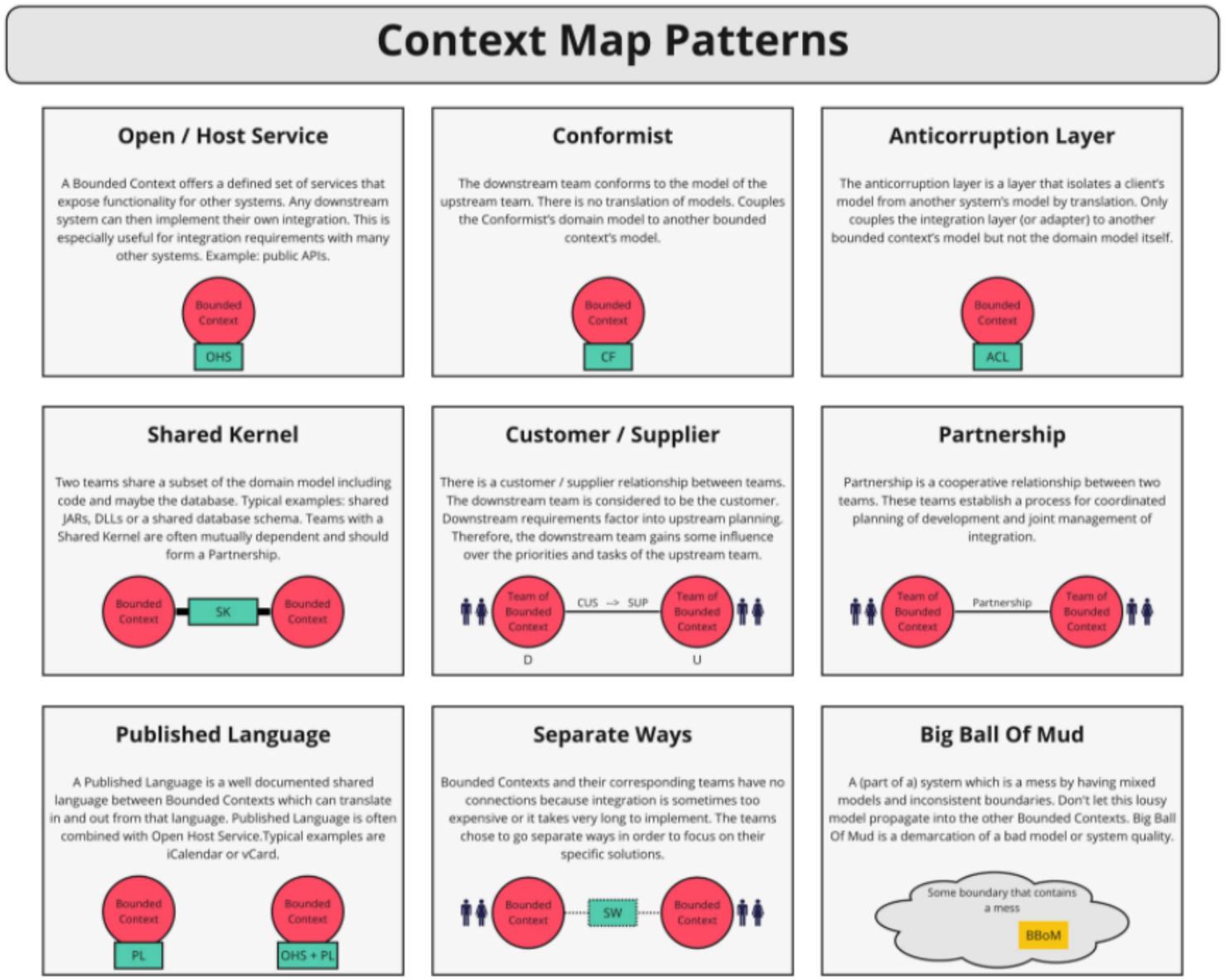
Flow of models between contexts



Context Map


Maintaining Model Integrity
keep model unified by
CONTINUOUS INTEGRATION
BOUNDED CONTEXT
SHARED KERNEL
CUSTOMER/ SUPPLIER TEAMS
names enter
assess/overview relationships with
CONTEXT MAP
overlap allied contexts through
relate allied contexts as
overlap unilaterally as
CONFORMIST
UBIQUITOUS LANGUAGE
segregate the conceptual messes
translate and insulate unilaterally with
free teams to go
support multiple clients through
OPEN HOST SERVICE
formalize as
PUBLISHED LANGUAGE
BIG BALL OF MUD
ANTICORRUPTION LAYER
SEPARATE WAYS

Context Map



https://github.com/ddd-crew/context-mapping

Team Relationships

https://github.com/ddd-crew/context-mapping

Problem space with Event Storming workshop

Event Storming

Event Storming

Event Storming

AddProductToBasket Add product AddProductToBasketEvent

Q/A