Catalyst

Uber’s Serverless Platform

Shawn Burke - Staff Engineer
Uber Seattle

Why Serverless?

- Complexity!
  - Microservices, Languages, Client Libs, Tools
  - Product teams have basic infrastructure needs
- Stable, consistent app platform pays huge dividends
- Abstraction => *Simplicity* => **Leverage**

Why are we building it?
(aka “Why don’t you just use AWS Lambda?”)

- Multi-cloud strategy
- Uber doesn’t run prod on AWS today
- Performance, extensibility requirements
  - New source types
  - Granular visibility/control
- Integration with existing systems

Catalyst Goals

- DEVELOPER FOCUSED: runs on desktop and in prod
- Dramatically simplify the process of delivering business logic
- Unify disparate systems with common patterns
- Pluggability: Multi-Language (handlers), extensible sources
- Minimal hard dependencies on underlying compute & networking
- **FAST**: < 5ms P99 Catalyst “tax”

End-to-End Experience

- Standardized service layout and execution
- Common coding model across sources types
- Batteries ARE Included: Config, Logging, Metrics, Dashboards, Crash Bucketing, Multi-Datacenter, Telemetry, Capture/Replay, Tracing
- Fun, fast iteration loop: code => test => debug => deploy

Writing Handlers: Go Example

```
type group struct {}
func (g *group) HelloWorldHandler(ctx context.Context, rw http.ResponseWriter, req *http.Request) {
      rw.Write([]byte(":-)"))
}

func MyKafkaHandler1(ctx context.C                                    r {
      catalyst.Logger(ctx).Info("[                                    ("len", len(msg)))
      return nil
}

func RegisterHandlers(catalyst *catalyst.Registry, g group) {
  catalyst.Register(chttp.Handler(g.HelloWorldHandler, "/", http.MethodGet))
  catalyst.Register(kafka.Handler(MyKafkaHandler1, "my-kafka-topic"))
}
```

```
logger := catalyst.Logger(ctx)
config := catalyst.Config(ctx)
metrics:= catalyst.Metrics(ctx)
```

Writing Handlers: Java

```java
@Group
public class Handlers {
   private static final Logger LOG = LoggerFactory.getLogger(Handlers.class);

   @Handler
   @HTTP(path="/", method=Method.GET)
   public CompletableFuture<String> index(Context ctx) {
       LOG.info("We just got a message from #{} ", ctx.userHeaders.get("User-Agent"));
       return CompletableFuture.completedFuture(":-)");
   }
   @Handler
   @Kafka(topic="my-kafka-topic", consumeGroup="my-cg")
   public void myTopicMessage(Context ctx, Payload payload) {
   }
}
```

Glossary

We have the best words.

- **Source:** Converts external events to Catalyst messages
- **Worker:** Binary containing user code
- **Handler**: Individual event handler
- **Group**: N handlers; unit of deployment

Architecture: Runtime/Data Plane

Nanny
- Goal State
- Process Lifecycle & Restart

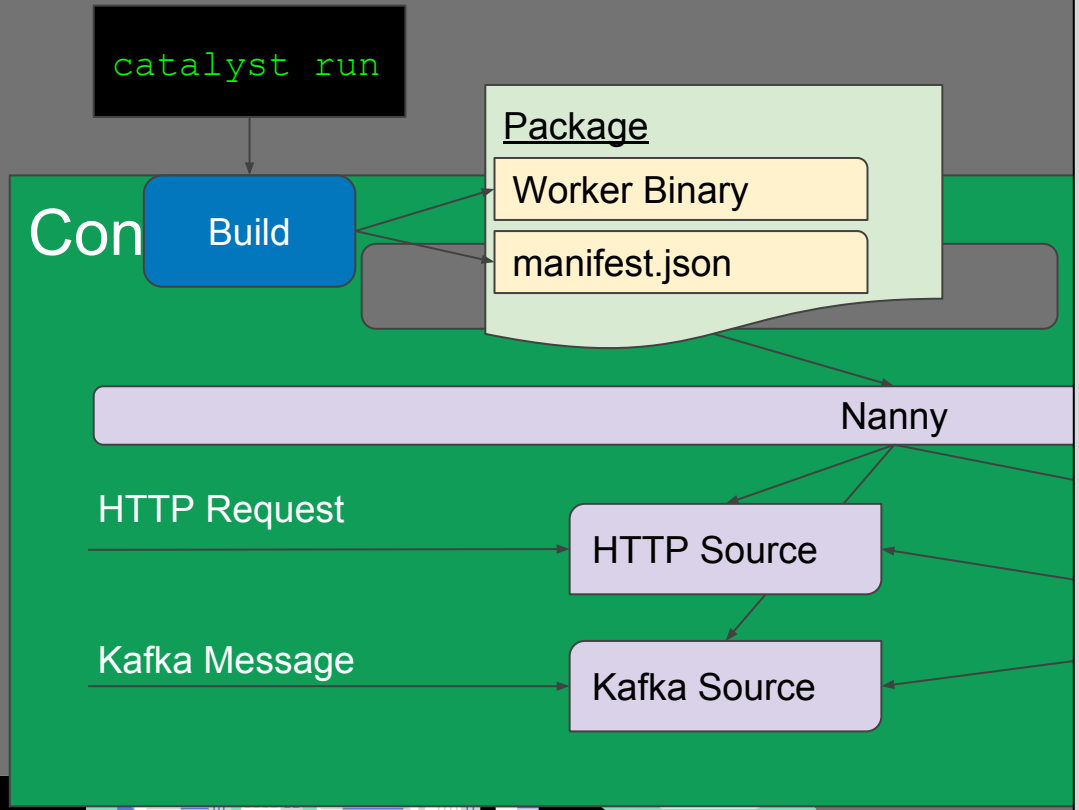
- Each box is a process
- Worker & Sources
  - Local: Worker from build
  - Production: Worker & Sources from S3
- **Shared Nothing Architecture**

Sources: Kafka, GRPC, HTTP,etc
- Per Event Type
- Flow Control
- Telemetry
- Capture/Replay

GRPC

Worker
- User Code
- Dispatching
- Telemetry
- Heartbeats

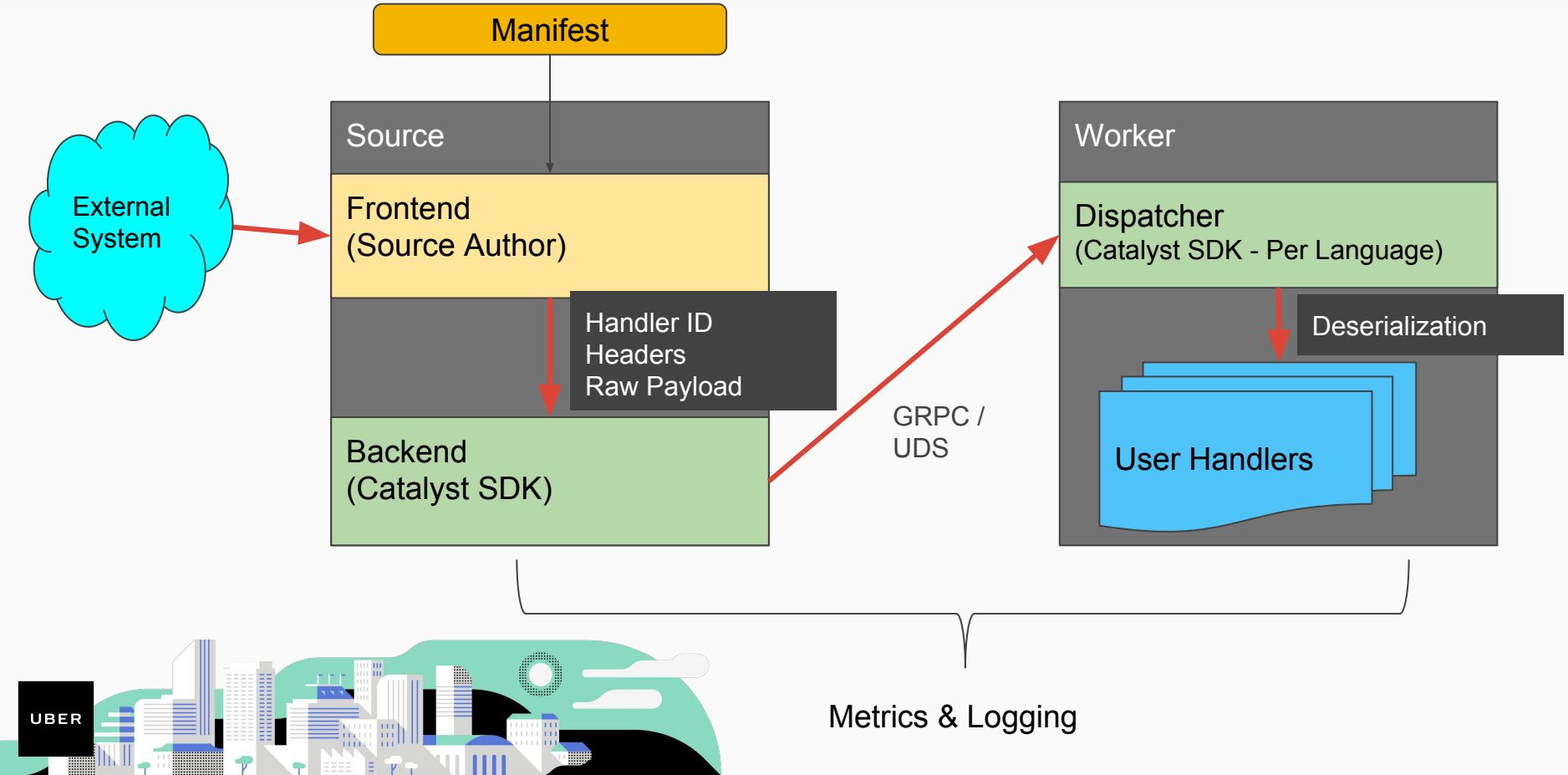
Running Handlers: Local Run



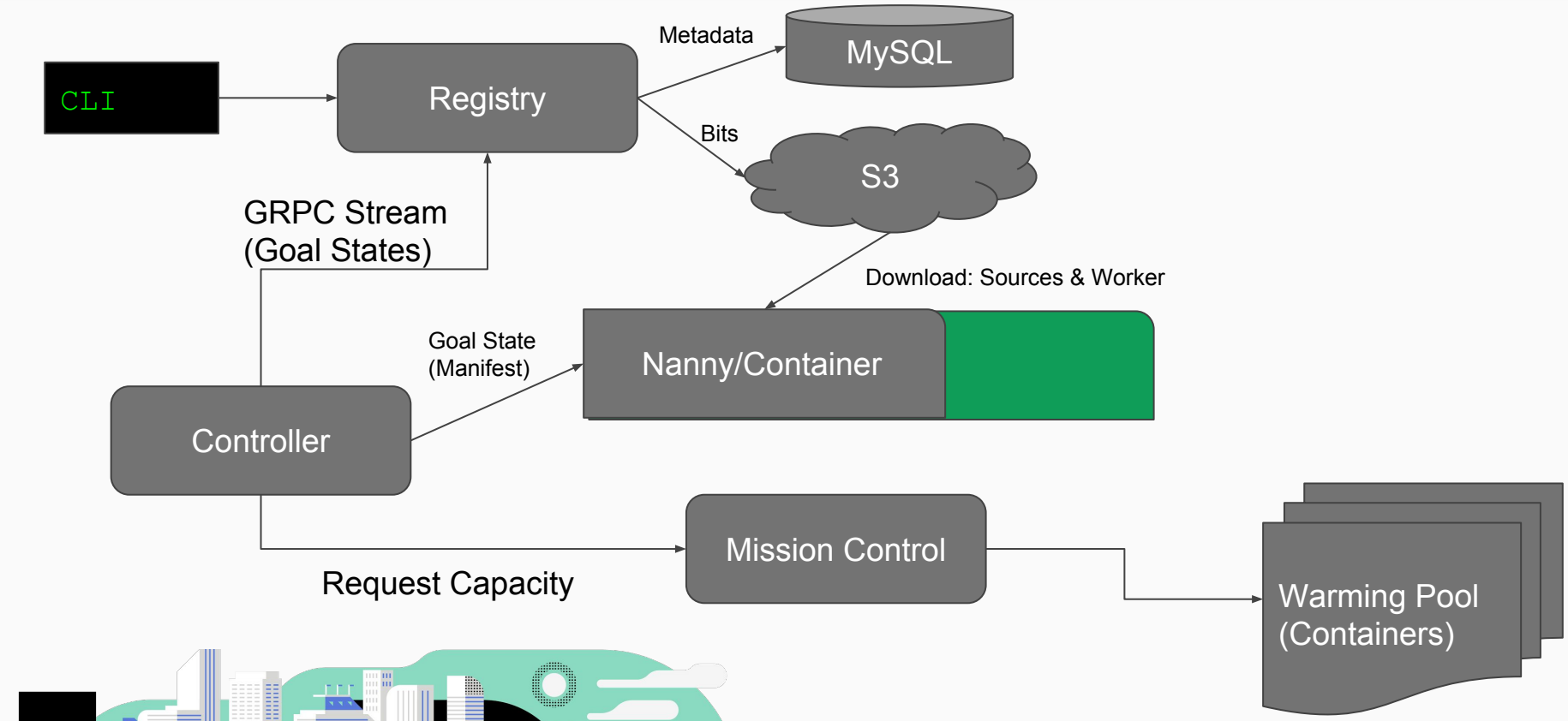
manifest.json

```
{
 "group_name": "my_http_demo",
 "project_name": "examples",
 "runtime": "go",
 "handlers": [
  {
   "name": "http_GET HelloWorldHandler",
   "source_type": "http",
   "params": {
    "method": "GET",
    "path": "/"
   },
   "config": {
    ...
   }
  },
  {
      "name": "kafka_MyKafkaHandler1",
      "source_type": "kafka",
      "params": {
        "offset": "newest",
        "topic": "my-kafka-topic",
      }
  }
 ],
 "built_at": "2017-02-08T14:38:36Z",
 "platform": "darwin"
}`
```

Anatomy of a Request



Architecture: Control Plane



Catalyst Today

- Written in Go
- Onboarding customers
- Persistent Containers, “Tax” P99 ~2ms
- Production hardening
  - Load testing
  - Testing negative scenarios and failure cases
- Bringing errors and information close to users

Catalyst Tomorrow

- Auto scaling
- Integrated (source-specific) status reporting
  - E.g. report status of underlying Kafka topics
- Advanced scenarios
  - Long-tail handlers
  - SLA-based priority
  - Traffic-based placement
- Goal: Open Source (no timeline quite yet)

Questions?

sburke@uber.com