DevOpsCon

Brett Smith
<xbcsmith@gmail.com>
20250020

DevOpsCon

Workshop: Building an Event-Driven CI/CD Provenance System

Brett Smith
<xbcsmith@gmail.com>
20250020

devmio NEW YORK WEEK

Event Driven CI/CD Workshop

View the workshop:

https://github.com/xbcsmith/epr-workshop

Clone the workshop:

git clone git@github.com:xbcsmith/epr-workshop.git

Session 1: Introduction to Event Driven Architecture

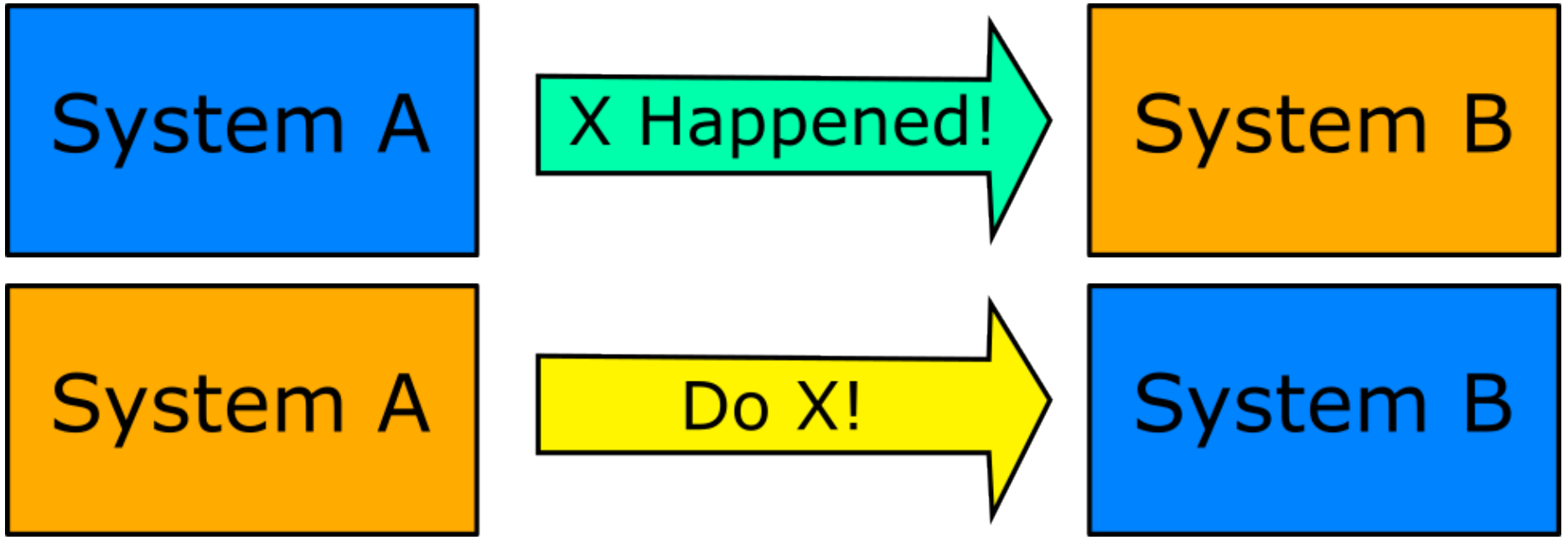
- Why use microservices in event driven architecture?
- What are Cloud Events?
- What are CDEvents?
- What is EPR and how does it fit in an event driven CI/CD system?

Introduction to Event Driven Microservice Architectures

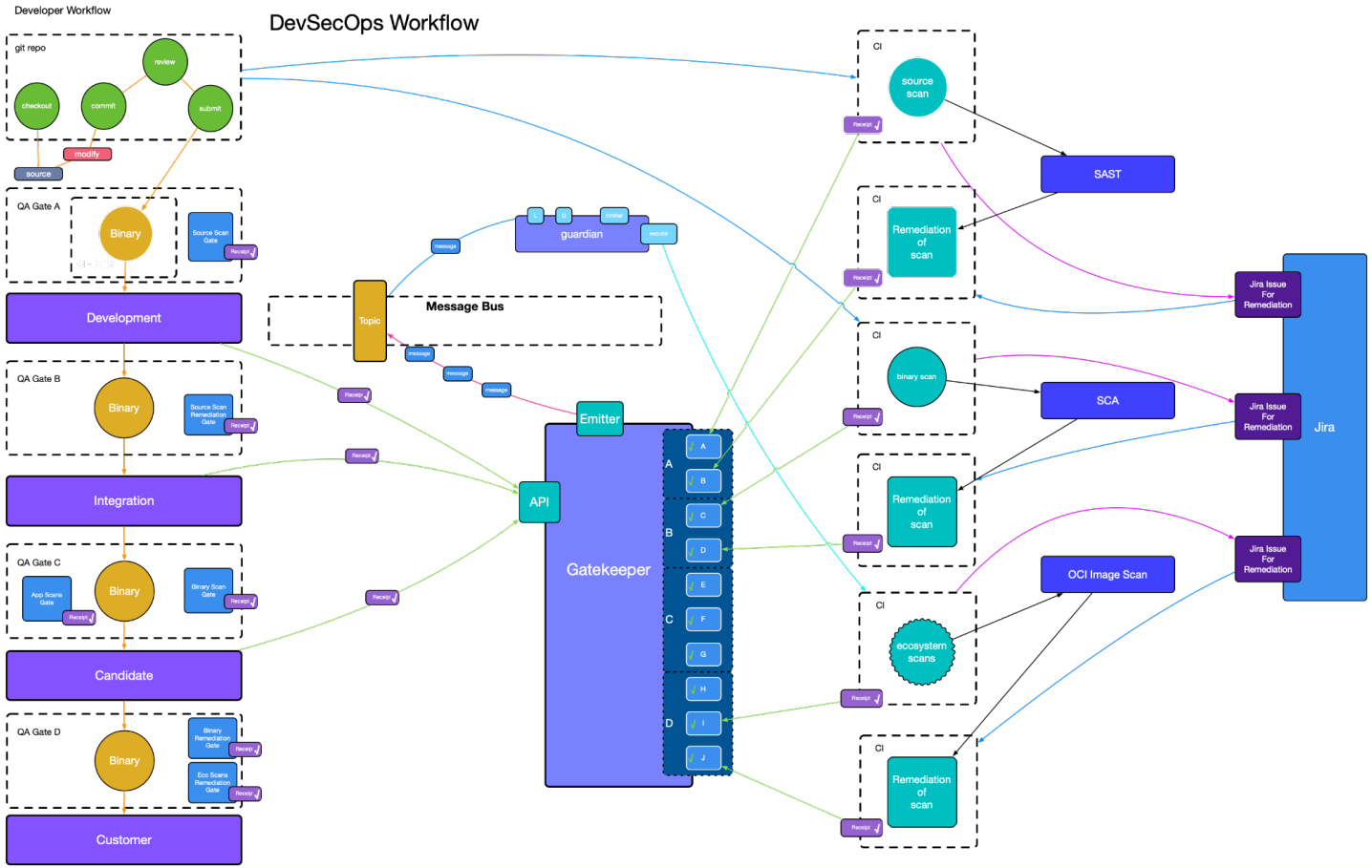
- Asynchronous
- Allows for scalability and modularity
- Provides an audit trail
- Services are specialized
- Independent SDLC

- Creeping system complexity
- Eventual consistency
- Increased maintenance
- Downstream flooding
- Dead Letters

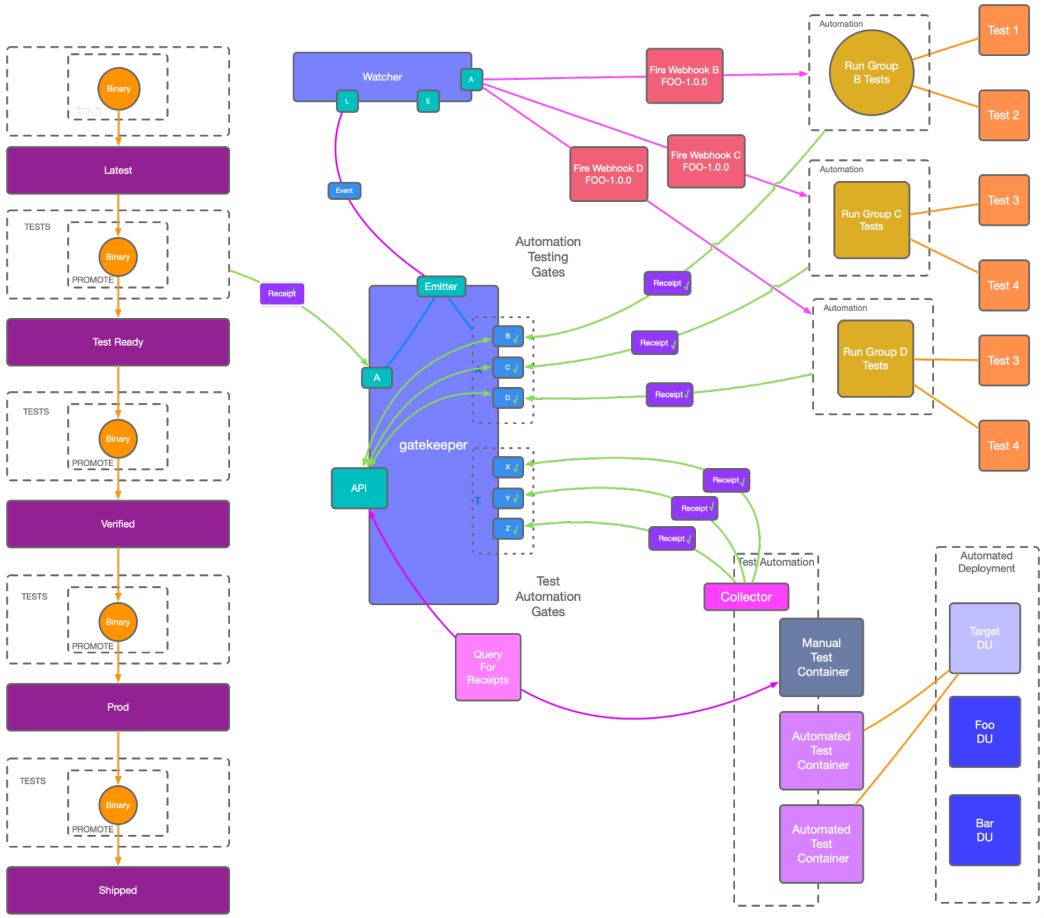
Declarative and Imperative Events



Real World


Developer Workflow
git repo
review
checkout
commit
submit
modify
source
QA Gate A
Binary
Source Scan Gate
Development
QA Gate B
Binary
Source Scan Remediation Gate
Integration
QA Gate C
App Scans Gate
Binary
Binary Scan Gate
Candidate
QA Gate D
Binary
Binary Remediation Gate
Eco Scans Remediation Gate
Customer
DevSecOps Workflow
guardian
Message Bus
Topic
Emitter
API
Gatekeeper
A
B
C
D
CI
source scan
Remediation of scan
binary scan
Remediation of scan
ecosystem scans
Remediation of scan
SAST
SCA
OCI Image Scan
Jira Issue For Remediation
Jira Issue For Remediation
Jira Issue For Remediation
Jira

Real World


Binary
Latest
TESTS
PROMOTE
Test Ready
Verified
Prod
Shipped
Watcher
Event
Emitter
Receipt
A
B
C
D
gatekeeper
API
X
Y
Z
Automation Testing Gates
Test Automation Gates
Query For Receipts
Fire Webhook B FOO-1.0.0
Fire Webhook C FOO-1.0.0
Fire Webhook D FOO-1.0.0
Automation
Run Group B Tests
Run Group C Tests
Run Group D Tests
Test 1
Test 2
Test 3
Test 4
Test Automation
Collector
Manual Test Container
Automated Test Container
Automated Deployment
Target DU
Foo DU
Bar DU

Cloud Events

- Interoperability
- Producer Consumer Independence
- Asynchronous Communication
- Standard Data Model

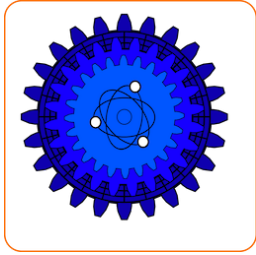
- Dynamic Event Interest Expression
- Flexible Event Consumption
- Consumer Evolution

CDEvents

- Common Vocabulary
- Normalized Form
- System Agnostic

- Declarative events
- Simplified Integration
- Targeted Notifications

Event Provenance Registry (EPR)



- Events
- Event Receivers
- Event Receiver Groups

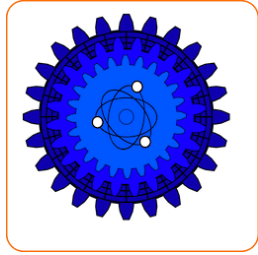
- Gating
- Auditing
- Provenance

- Messages
- CloudEvents Spec
- Declarative events
- Imperative events
- Watchers

Session 2: Introduction to Event Provenance Registry

- Setup and deploy Event Provenance Registry (EPR) server locally.
- Create a microservice to interact with EPR and events.
- Overview of the EPR Python SDK with examples.


EPR
Workshop



Setting Up the Environment

Docker – https://docs.docker.com/get-docker

Docker Compose – https://docs.docker.com/compose/install

Golang – https://go.dev/doc/install

Python – https://www.python.org/downloads

Git – https://git-scm.com/downloads

Redpanda (Kafka): Message Queue Overview

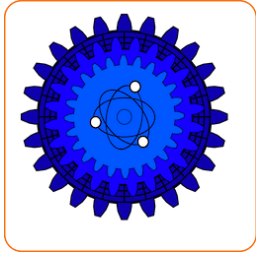
Redpanda is a Kafka-compatible event streaming platform.

It provides a message bus for our event-driven architecture.

Simple. Kafka® API-compatible. ZooKeeper® free. JVM free.

- Redpanda Nodes - Self contained service
- Redpanda Keeper - CLI for managing clusters
- Redpanda Console - UI for managing clusters

Redpanda (Kafka): Message Queue Overview



Terminal 1 : run the command to consume the message

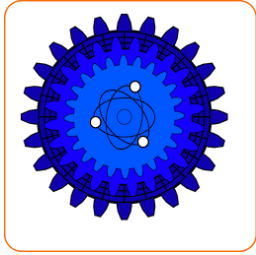
```
docker compose -f ./compose/docker-compose.yaml up
```

Terminal 2 : run the commands create a topic and produce the message

```
docker exec -it redpanda rpk topic create epr.dev.events --brokers=localhost:9092
docker exec -it redpanda rpk topic produce epr.dev.events --brokers=localhost:9092
```

Terminal 3 : run the command to consume the message

```
docker exec -it redpanda rpk topic consume epr.dev.events --brokers=localhost:9092
```



EPR Concepts

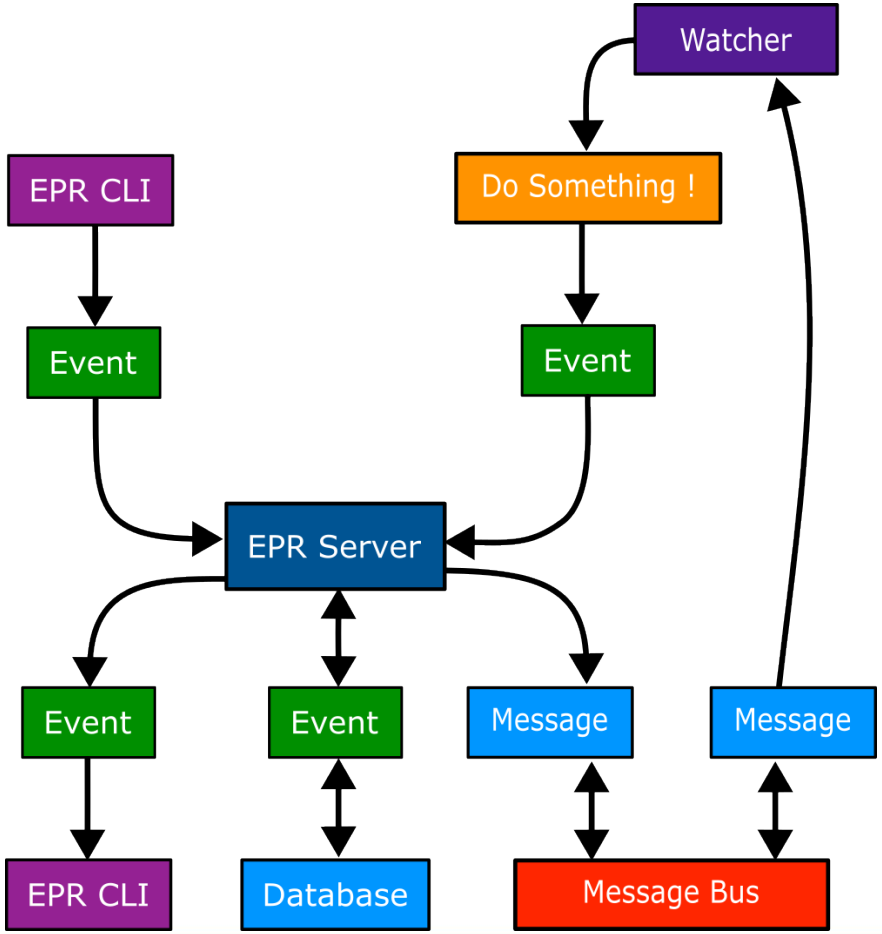
- Events
- Event Receivers
- Event Receiver Groups

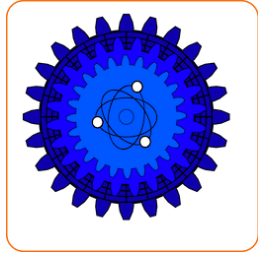
- Gating
- Auditing
- Provenance

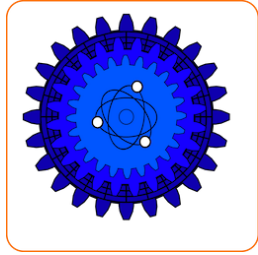
- N - Name
- V - Version
- R - Release
- P - Platform ID
- P - Package

- Messages
- CloudEvents Spec
- Declarative events
- Imperative events
- Watchers

EPR Workflow



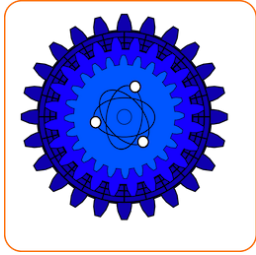




Exploring EPR: Codebase Overview

Use the following command to clone the EPR project repository:

```
git clone git@github.com:sassoftware/event-provenance-registry.git
```



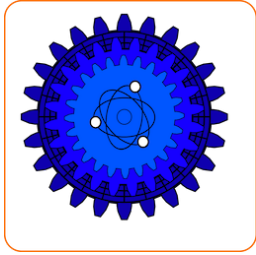
Exploring EPR: Start the Backend

Open a new terminal and cd to the event-provenance-registry repository.

```
cd event-provenance-registry
```

Use the following command to start the backend services for EPR:

```
docker compose -f ./docker-compose.services.yaml up
```



Building EPR: Service

Open a new terminal and cd to the event-provenance-registry repository.

```
cd event-provenance-registry
```

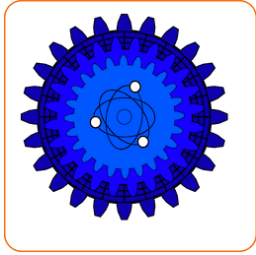
Export the following environment variables:

```
export EPR_TOPIC=epr.dev.events
export EPR_BROKERS=localhost:19092
export EPR_DB=postgres://localhost:5432
```

Then we can start the EPR server with the following command:

```
go run main.go
```

The server will be available on localhost:8042



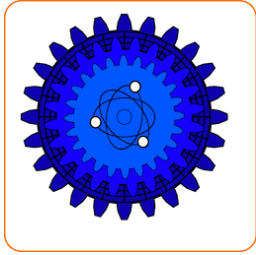
Setup EPR: Workflow

EPR workflow to setup the server to produce events is as follows:

1. Create an Event Receiver.
2. Post an Event to the Event Receiver.
3. EPR produces message on topic.

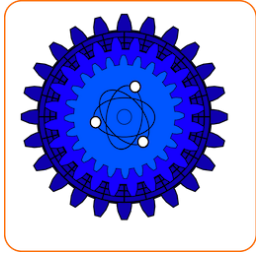
To leverage Event Receiver Groups we would follow this workflow:

1. Create several Event Receivers
2. Create an Event Receiver Group with all the Event Receivers.
3. Create an Event with identical NVRPP for each Event Receiver in the Group.
4. EPR produces a message for each Event on the topic.
5. When the last Event is sent EPR produces a message for the Event Receiver Group on the topic.



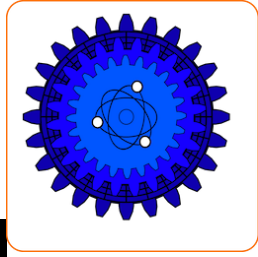
Curl EPR: Create an Event Receiver

```
curl --location --request POST 'http://localhost:8042/api/v1/receivers' \
--header 'Content-Type: application/json' \
--data-raw '{
 "name": "foobar",
 "type": "foo.bar",
 "version": "1.1.3",
 "description": "The event receiver of Brixton",
 "schema": {
   "type": "object",
   "properties": {
     "name": {
       "type": "string"
     }
   }
 }
}'
```



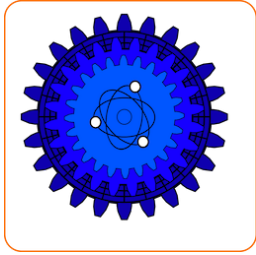
Curl EPR: Create an Event

```
curl --location --request POST 'http://localhost:8042/api/v1/events' \
--header 'Content-Type: application/json' \
--data-raw '{
 "name": "magnificent",
 "version": "7.0.1",
 "release": "2023.11.16",
 "platform_id": "linux",
 "package": "docker",
 "description": "blah",
 "payload": {
   "name": "joe"
 },
 "success": true,
 "event_receiver_id": "<PASTE EVENT RECEIVER ID FROM FIRST CURL COMMAND>"
}
```



Curl EPR: Create an Event Receiver Group

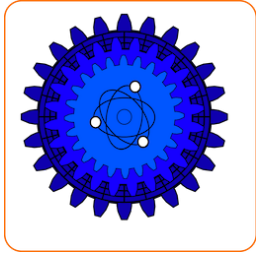
```
curl --location --request POST 'http://localhost:8042/api/v1/groups'
\
--header 'Content-Type: application/json' \
--data-raw '{
 "name": "the_clash",
 "type": "foo.bar",
 "version": "3.3.3",
 "description": "The only event receiver group that matters",
 "enabled": true,
 "event_receiver_ids": [
   "PASTE EVENT RECEIVER ID FROM FIRST CURL COMMAND"
 ]
}
```



The GraphQL Playground is a tool that allows you to test your GraphQL queries in a browser.

The graphql playground will now be accessible at:

`http://localhost:8042/api/v1/graphql`

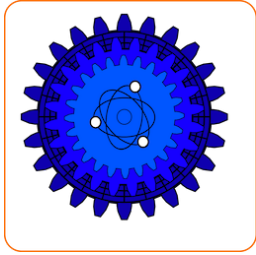


Using EPR: GraphQL

Create an Event Receiver by pasting the mutation into the GraphQL Playground.

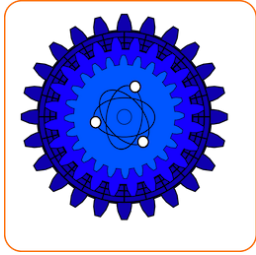
```
mutation {
 create_event_receiver(
   event_receiver: {
     name: "the_clash"
     version: "1.0.0"
     type: "london.calling"
     description: "The only band that matters"
     schema: "{\"name\": \"value\"}"
   }
 )
}
```

Using EPR: GraphQL



Create an Event Receiver Group by pasting the mutation into the GraphQL Playground. Use the ID from the last command for the event_receivers_ids field.

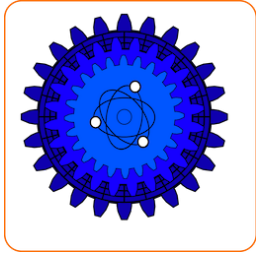
```
mutation {
 create_event_receiver_group(
   event_receiver_group: {
     name: "foobar"
     version: "1.0.0"
     description: "a fake event receiver group"
     enabled: true
     event_receiver_ids: ["ID_RETURNED_FROM_PREVIOUS_MUTATION"]
     type: "test.test.test"
   }
 )
}
```



Using EPR: GraphQL

Create an Event by pasting the mutation into the GraphQL Playground.

```
mutation {
 create_event(
   event: {
     name: "foo"
     version: "1.0.0"
     release: "20231103"
     platform_id: "platformID"
     package: "package"
     description: "The Foo of Brixton"
     payload: "{\"name\": \"value\"}"
     event_receiver_id: "ID_RETURNED_FROM_PREVIOUS_MUTATION"
     success: true
   }
 )
}
```



Building EPR: CLI

Open a new terminal and cd to the cli directory in the event-provenance-registry repository.

```
cd event-provenance-registry/cli
```

Build CLI with the following commands:

Linux

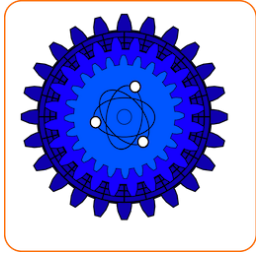
```
make PREFIX=$(go env GOPATH) install
```

Mac OS X M1

```
make PREFIX=$(go env GOPATH) install-darwin-arm64
```

Export the following environment variables:

```
export EPR_TOPIC=epr.dev.events
export EPR_BROKERS=localhost:19092
export EPR_DB=postgres://localhost:5432
```

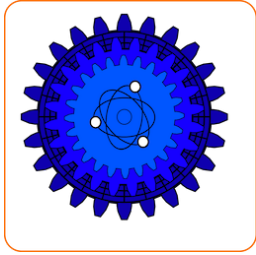


Using EPR: CLI

Export the following environment variables:

```
export EPR_TOPIC=epr.dev.events
export EPR_BROKERS=localhost:19092
export EPR_DB=postgres://localhost:5432
```

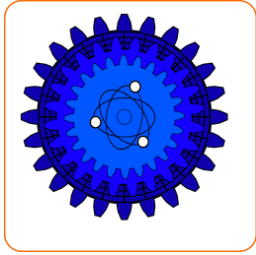
Using EPR: CLI



Create a pair of Event Receivers

```
epr-cli receiver create \
     --name "foo-cli" \
     --version "1.0.0" \
     --description "foo cli created foo" \
     --type "epr.foo.cli" \
     --schema "{}"
```

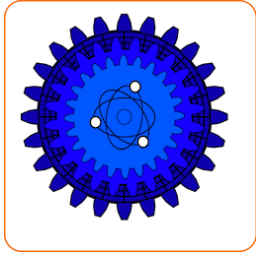
```
epr-cli receiver create \
     --name "bar-cli" \
     --version "1.0.0" \
     --description "bar cli created bar" \
     --type "epr.bar.cli" \
     --schema "{}"
```



Using EPR: CLI

Create an Event Receiver Group. Use the IDs from the Event Receivers

```
epr-cli group create \
     --name "foo-group-cli" \
     --version "1.0.0" \
     --description "foo cli created foo group" \
     --type "epr.foo.group.cli" \
     --event-receiver-ids "01HKX0J9KS8AASMRYX61458N41 01HKX0KY3B31MR3XKJWTDZ4EQ0"
```



Using EPR: CLI

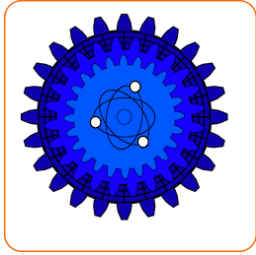
Search for objects by ID, or Name and Version, or Type.

```
epr-cli receiver search --id 01HKX0KY3B31MR3XKJWTDZ4EQ0 --fields all

epr-cli receiver search --type epr.foo.cli --fields all

epr-cli receiver search --name foo-cli --version 1.0.0 --fields all
```

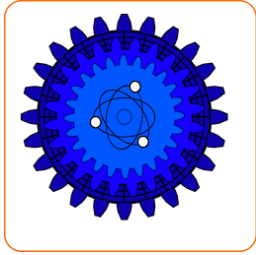
Extended EPR: Creating a watcher



Use the EPR watcher SDK to craft a watcher, which actively listens for events originating from the EPR server.

Make a new directory for your watcher and create a `main.go` in that directory.

```
mkdir foo
cd foo
touch main.go
```



Extended EPR: Creating a watcher

Now open the `main.go` in your favorite editor (Vim).

Add the following code:

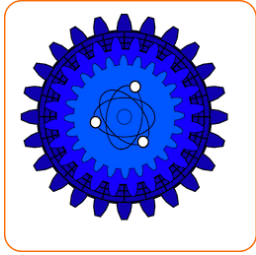
```
package main

import (
"encoding/json"
"log"

"github.com/sassoftware/event-provenance-registry/pkg/message"
"github.com/sassoftware/event-provenance-registry/pkg/watcher"
)

func main() {
seeds := []string{"localhost:19092"}
topics := []string{"epr.dev.events"}
consumerGroup := "watcher-workshop"
watcher, err := watcher.New(seeds, topics, consumerGroup)
if err != nil {
 panic(err)
}
defer watcher.Client.Close()
go watcher.StartTaskHandler(customTaskHandler)
watcher.ConsumeRecords(customMatcher)
}
```

Extended EPR: Creating a watcher



Add:

- customMatcher
  - matching the message type value.
- customTaskHandler
  - prints a message when a match is found.

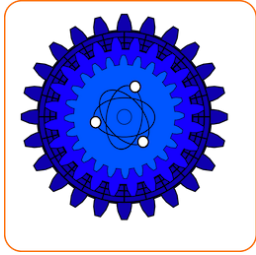
```
func customMatcher(msg *message.Message) bool {
return msg.Type == "foo.bar"
}

func customTaskHandler(msg *message.Message) error {
log.Default().Printf("I received a task with value '%v'", msg)
return nil
}
```

Save the file and run `go mod init` in your terminal.

Run `go mod tidy` to fill in the dependencies.

Extended EPR: Creating a watcher



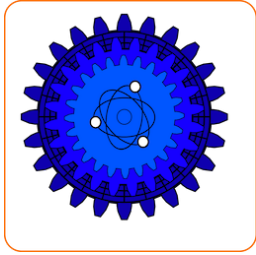
Start up the watcher and start consuming messages with the following command:

```
go run main.go
```

In a second terminal create an Event Receiver where type matches what you put in the customMatcher.

```
curl --location --request POST 'http://localhost:8042/api/v1/receivers' \
--header 'Content-Type: application/json' \
--data-raw '{
 "name": "watcher-workshop",
 "type": "foo.bar",
 "version": "1.0.0",
 "description": "The event receiver of Brixton",
 "enabled": true,
 "schema": {
 "type": "object",
 "properties": {
   "name": {
     "type": "string"
   }
 }
}
}'
```

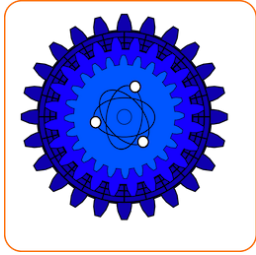
Extended EPR: Creating a watcher



Now create an Event.

```
curl --location --request POST 'http://localhost:8042/api/v1/events' \
--header 'Content-Type: application/json' \
--data-raw '{
   "name": "magnificent",
   "version": "7.0.1",
   "release": "2023.11.16",
   "platform_id": "linux",
   "package": "docker",
   "description": "blah",
   "payload": {"name":"joe"},
   "success": true,
   "event_receiver_id": "<PASTE EVENT RECEIVER ID FROM FIRST CURL COMMAND>"
}'
```

EPR: Python



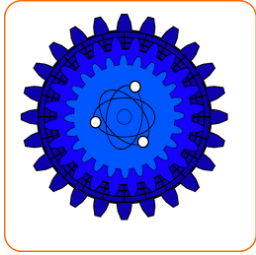
In this section we will walk through the development and usage of the EPR Python client.

- EPR Python Client
- Overview
- Development
- Model Example
- Client Create Example
- Create an event receiver
- Create an event receiver group
- Create an event
- EPR Python CLI

The commands in this section are long. Please refer to section 09-epr-python

Session 3: CDEvents and SBOMs

- Create and store CDEvents in our new ecosystem.
- Discuss how to leverage CDEvents and events in general.
- Use EPR to store and retrieve SBOMs.



EPR: SBOMs

In this section of the workshop, we will introduce the SBOMs.

- Event Receiver Schema for SBOMs
- CycloneDX bom schema
- Create a source SBOM event
- Create an SBOM for an OCI image

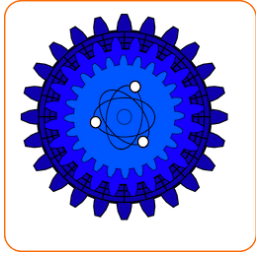
The commands in this section are long. Please refer to section 08-epr-sboms

EPR:CloudEvents

In this section we will cover the CloudEvents and EPR concepts and how to use them.

- Overview of CloudEvents
- Explore CloudEvents Data Structure
- Create CloudEvents
- Use CloudEvents with EPR

The commands in this section are long. Please refer to section 09-epr-cloudevents



EPR: CDEvents

In this section we will cover the CDEvents and EPR concepts and how to use them.

- Overview of CDEvents
- Explore CDEvents Data Structure
- Simulate Pipeline Activities with CDEvents
- EPR Integration with CDEvvents

The commands in this section are long. Please refer to section 10-epr-cdevents

EPR:CDViz

In this section we will cover the CDViz and CDEvents concepts and how to use them.

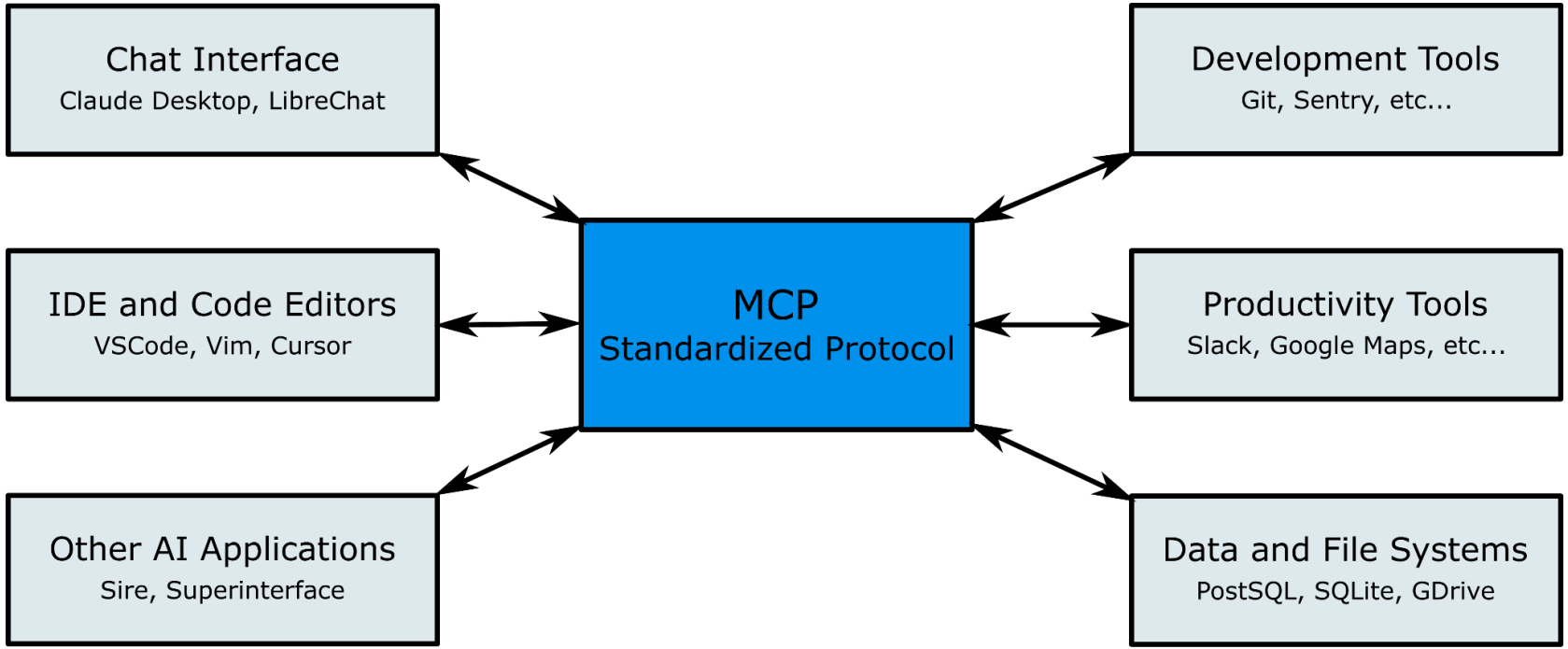
- Deploy CDViz
- Send CDEvents to CDViz
- Explore CDViz Dashboards and Capabilities

The commands in this section are long. Please refer to section 10.5-epr-cdviz

Session 4: EPR Agents and MCP Servers

- Overview of MCP servers and how they can be used in event driven systems.
- Introduction to the EPR MCP Server.
- Live Code MCP Server
- Expand our microservices to do more things.
- Wrap up.

EPR: Agents and MCP Servers



Model Context Protocol (MCP)

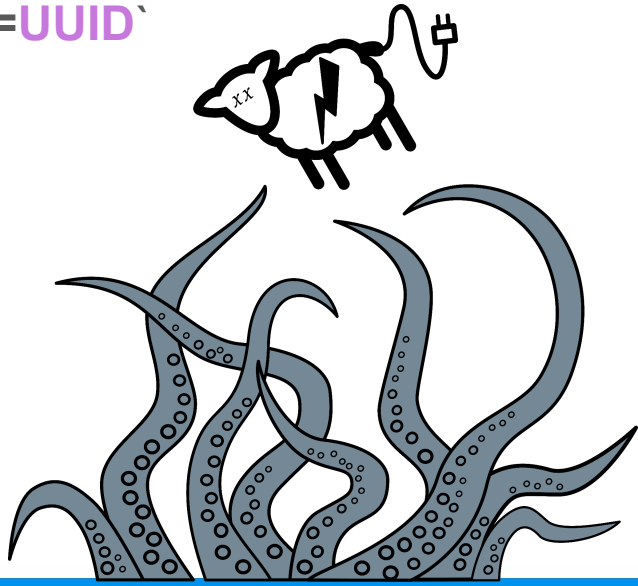
AI Assistant (Client) <=> MCP Server <=> API/Tools/Data

- Open standard introduced November 2024
- JSON-RPC interface over HTTP/stdio
- Standardized discovery for AI-tool integration
- No custom plugins required

MCP: Not So Secure by Design

Fundamental Design Flaws:

- Session IDs in URLs `**GET /messages/?sessionId=UUID**`
- Optional authentication standards
- Missing message integrity controls
- Trust model assumes good actors



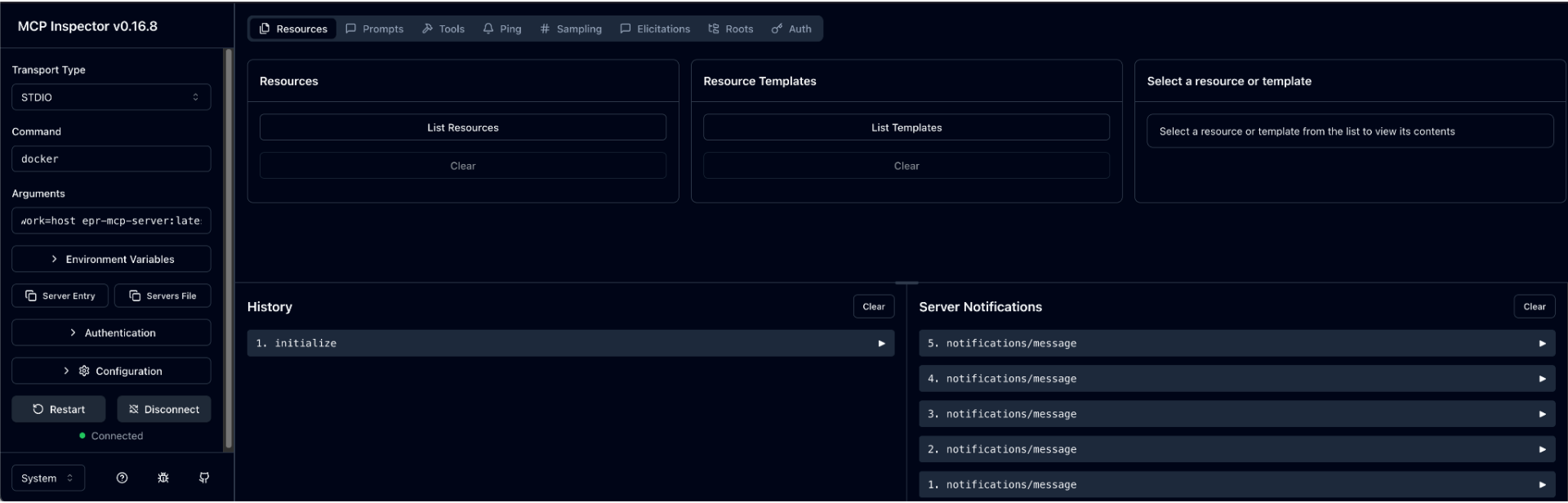
EPR: MCP Server

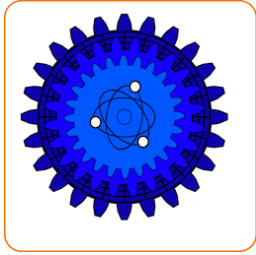
An MCP server (Model Context Protocol server) acts as a bridge between AI models and external tools, data sources, or APIs.

- Live Code an MCP Server

The commands in this section are long. Please refer to section 11-epr-mcp-server

EPR: MCP Inspector



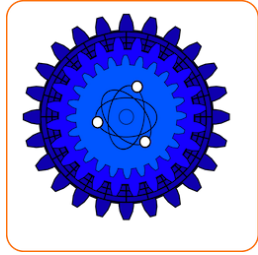


EPR: Workshop Challenge

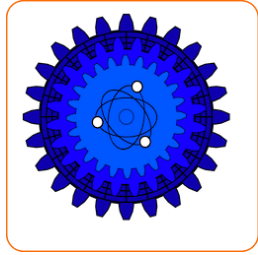
In this section we will expand on our EPR concepts we have learned in the previous sections. This is a free form exercise to expand on what we have learned.

Suggestions:

- Add a watcher to fire a webhook.
- Add a watcher to create a CDEvent.
- Add a watcher to create a SBOM.
- Add a Python watcher to create an Event.



AMA

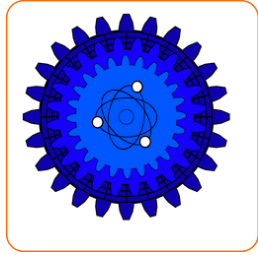


EPR Workshop

EPR

EPR-Python

Brett Smith
xbcsmith@gmail.com
20240405



I am Smitty and I am afraid of robots

Brett Smith
GitHub <https://github.com/xbcsmith>

Please,
don't forget
to vote!

devmio