Beyond REST and CRUD

Integration patterns in Microservices

Erin Schnabel, Red Hat · github.com/ebullient · www.ebullient.dev

Zineb Bendhiba, Red Hat · github.com/zbendhiba · zinebbendhiba.com

Essential factors

Autonomous
Portable
Disposable

I. Codebase
One codebase tracked in revision control, many deploys

II. Dependencies
Explicitly declare and isolate dependencies

III. Config
Store config in the environment

IV. Backing services
Treat backing services as attached resources

V. Build, release, run
Strictly separate build and run stages

VI. Processes
Execute the app as one or more stateless processes

VII. Port binding
Export services via port binding

VIII. Concurrency
Scale out via the process model

IX. Disposability
Maximize robustness with fast startup and graceful shutdown

X. Dev/prod parity
Keep development, staging, and production as similar as possible

XI. Logs
Treat logs as event streams

XII. Admin processes
Run admin/management tasks as one-off processes

Essential factors

Autonomous
Portable
Disposable
Independent

I. Codebase
One codebase tracked in revision control, many deploys

II. Dependencies
Explicitly declare and isolate dependencies

III. Config
Store config in the environment

IV. Backing services
Treat backing services as attached resources

V. Build, release, run
Strictly separate build and run stages

VI. Processes
Execute the app as one or more stateless processes

VII. Port binding
Export services via port binding

VIII. Concurrency
Scale out via the process model

IX. Disposability
Maximize robustness with fast startup and graceful shutdown

X. Dev/prod parity
Keep development, staging, and production as similar as possible

XI. Logs
Treat logs as event streams

XII. Admin processes
Run admin/management tasks as one-off processes

How do we adapt?

- Applications must evolve

How do we adapt?

- Applications must evolve
- Rarely ever a clean slate

How do we adapt?

- Applications must evolve
- Rarely ever a clean slate
- Decouple elements of the system

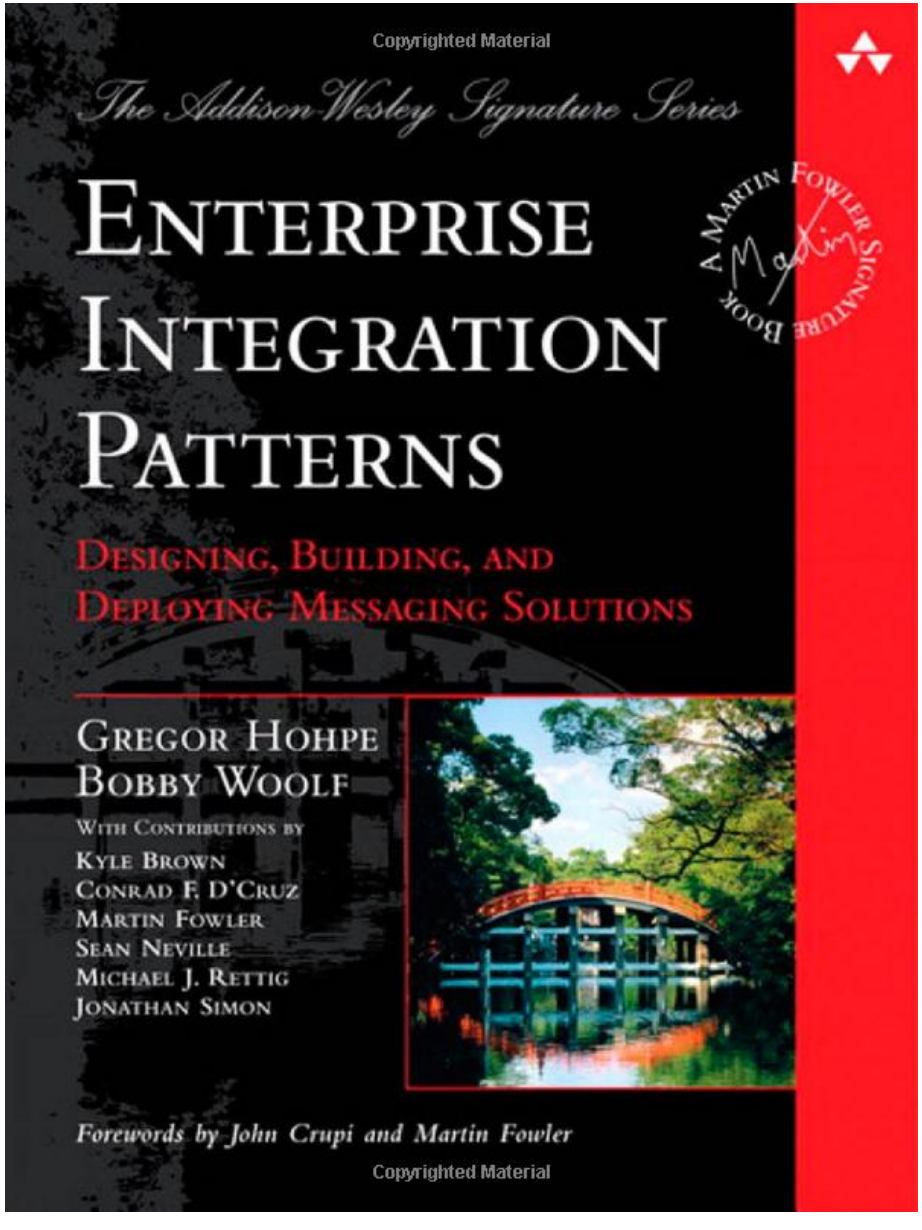
Systems Integration

Different perspective

Systems Integration

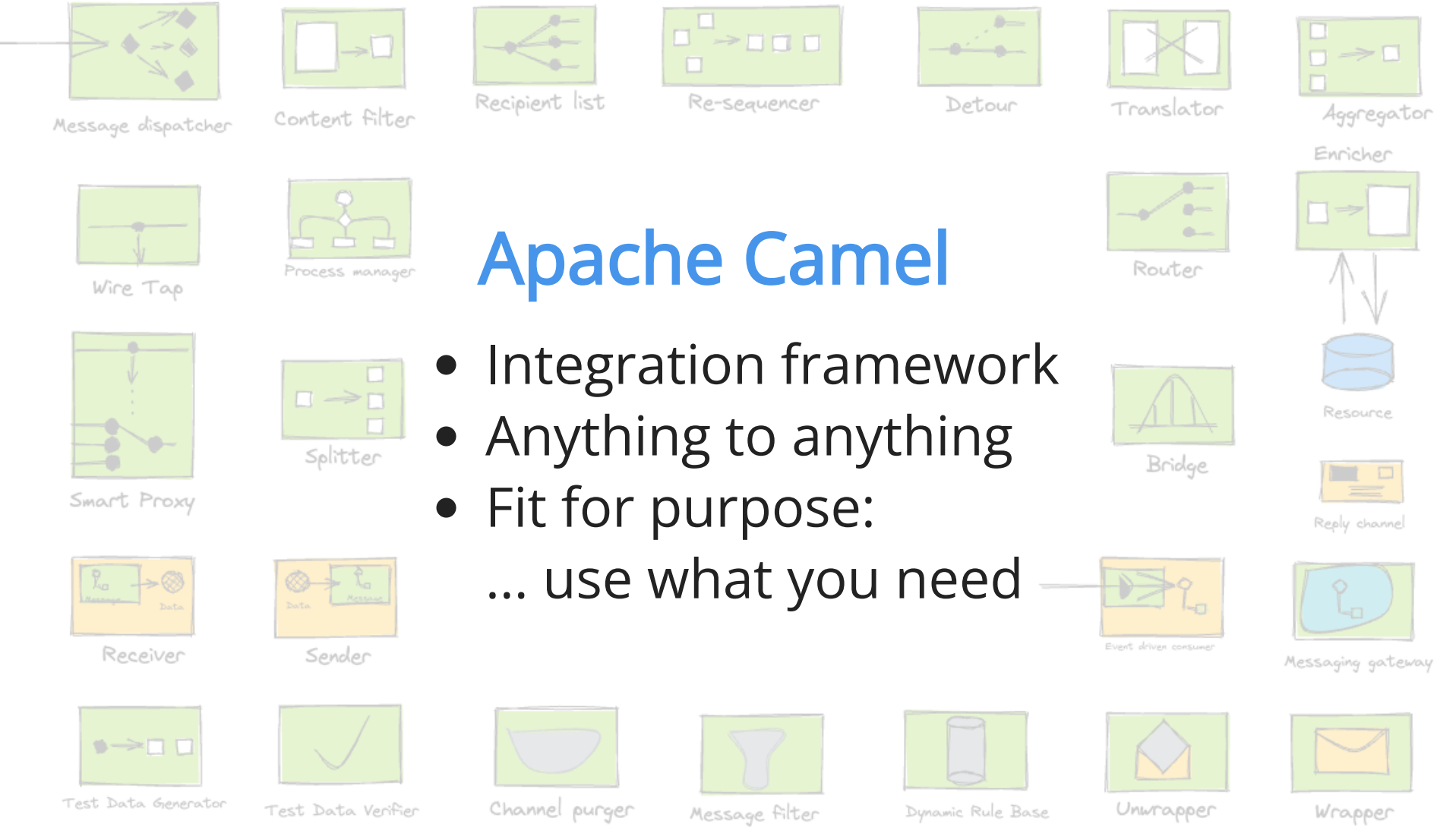
Different perspective

Message Routing



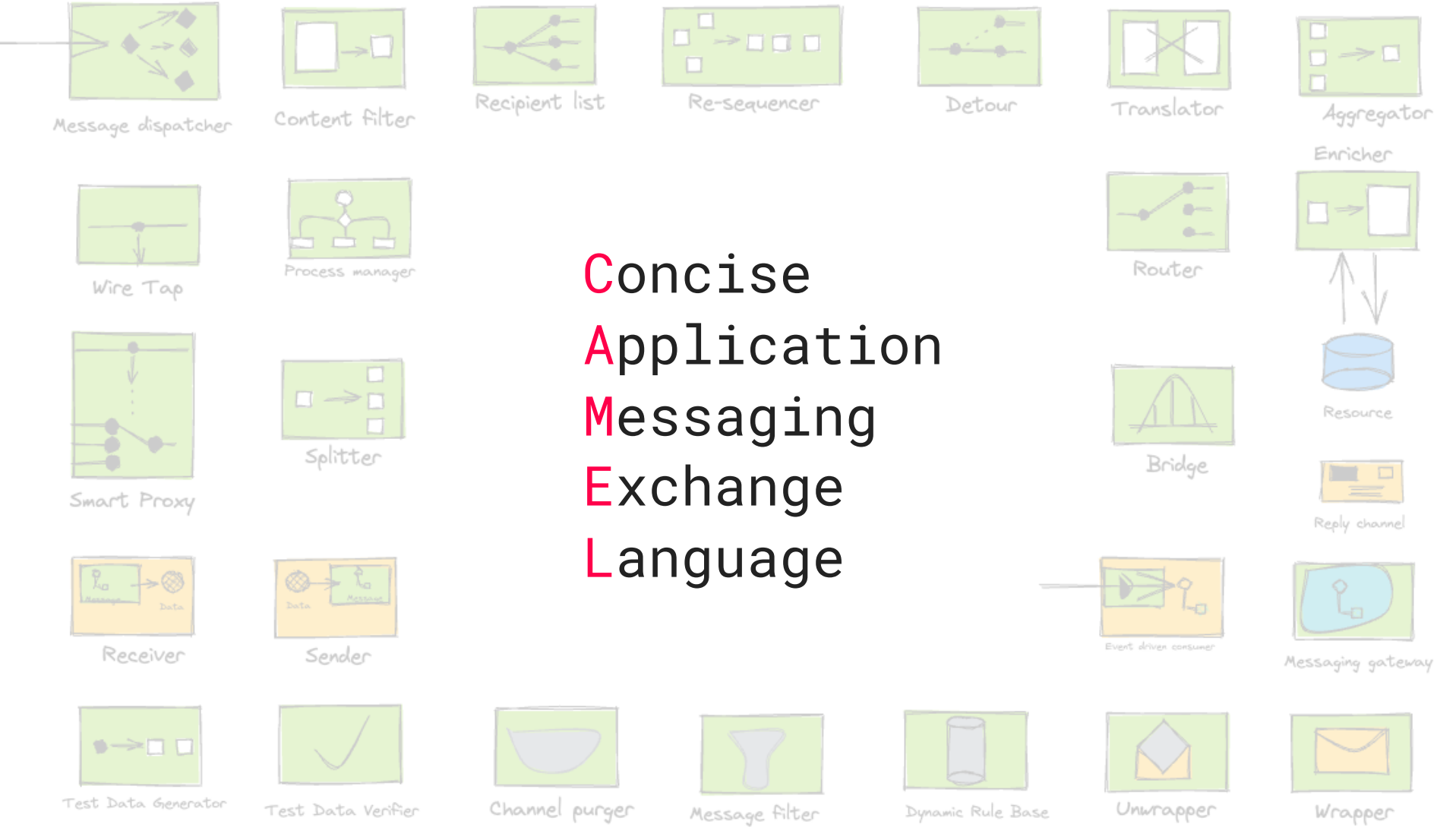
Enterprise patterns

Emergent
"Design patterns" Best practices



Apache Camel

- Integration framework
- Anything to anything
- Fit for purpose:
  … use what you need



Concise
Application
Messaging
Exchange
Language

Camel message routing...

Term	Meaning
`Message`	*data* transferred by a route
`Exchange`	*envelope*; wraps the data
`Endpoint`	a channel; *receiver* or *sender*
`Component`	know-how; creates endpoints
`Processor`	Java API; *custom logic*

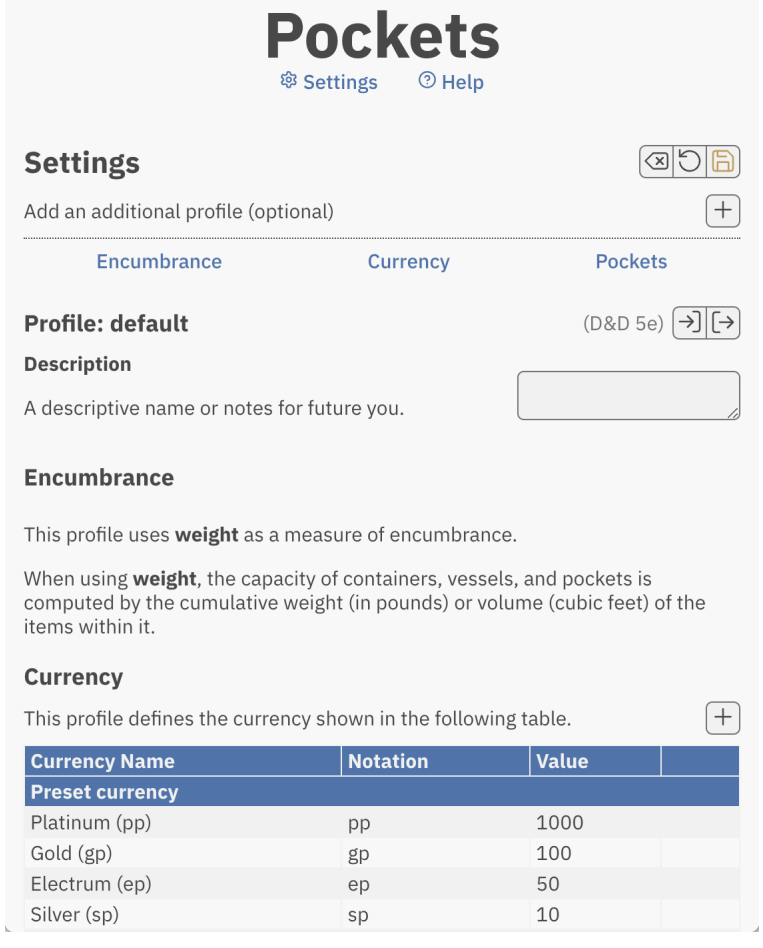
Pockets

- Inventory management
- Command-line
- local-only Web UI

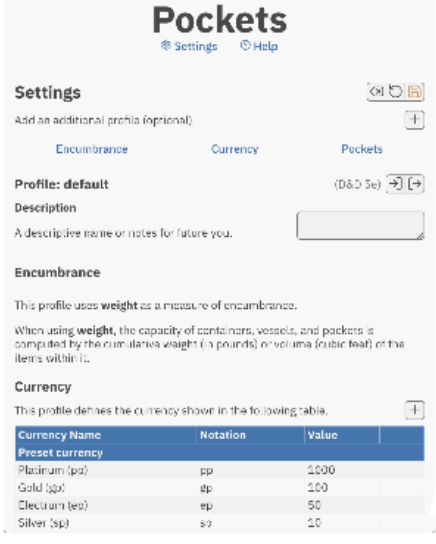
```
What have you got in your pockets?

Usage: pockets [-hVi] [--debug] [--quiet] [COM
Options:
      --debug          Enable debug output
      --quiet          Enable quieter (less ver
  -i, --interactive    Confirm and prompt for m
  -h, --help           Show this help message a
  -V, --version        Print version informatio

Commands:
  add                    🎁 Add items to a pocket
  create                 🎁 Create an item or poc
  buy                    💰 Exchange currency for
  ls, list               🔎 Look at your stuff.
  tx, move, trade        🚚 Move or trade items.
  rm, consume, remove    💸 Remove pockets, items
  profile                🏠 Add, update, or delet
  reset                  🔥 Reset everything.

🔥 indicates destructive commands and options.

To backup your configuration and data:
  'pockets profile export --data <profile name
```

Pockets: The plan

Pockets: The plan

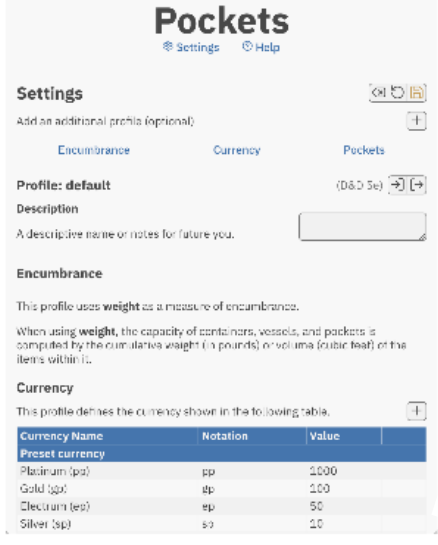


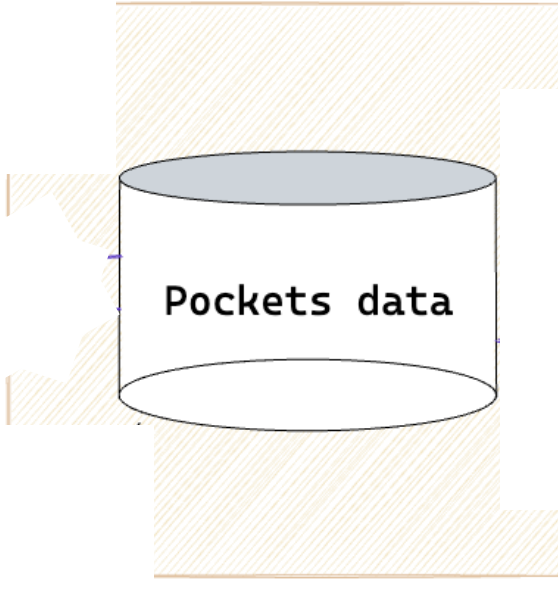
Pockets: The plan




```
> pockets ...
```

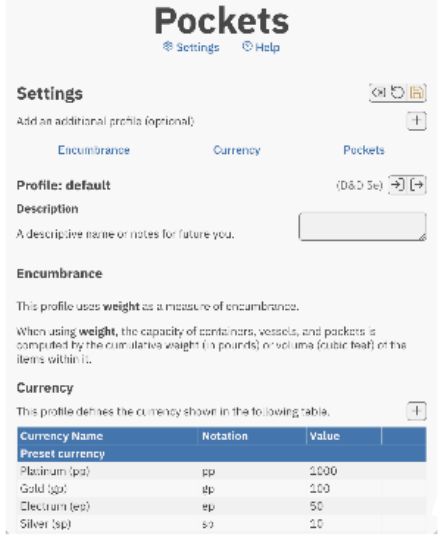
Pockets: The plan

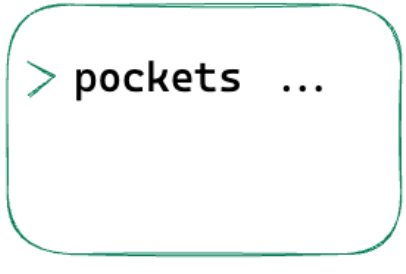


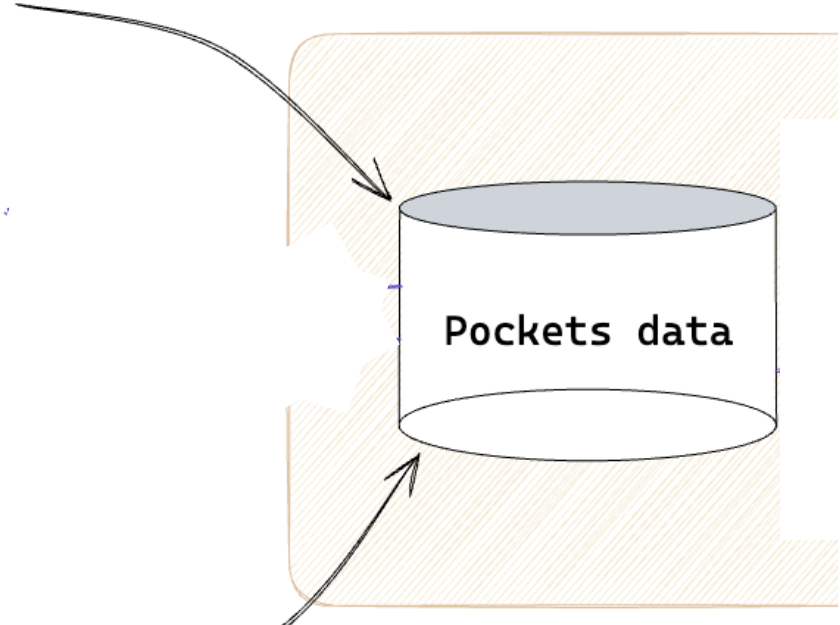



```
> pockets ...
```

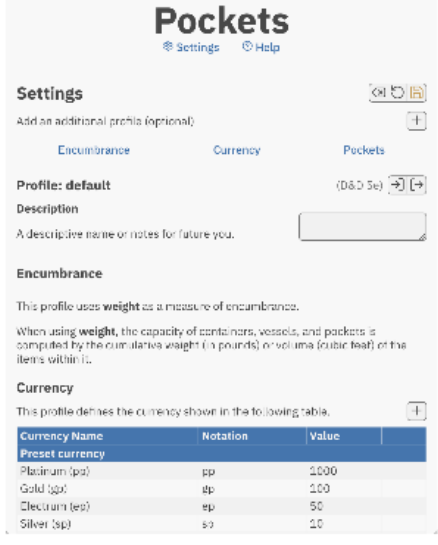
Pockets: The plan



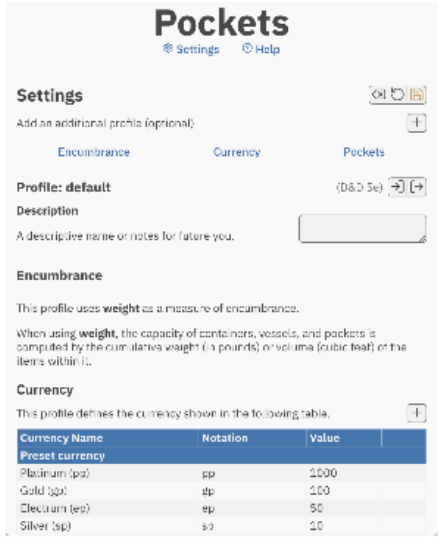




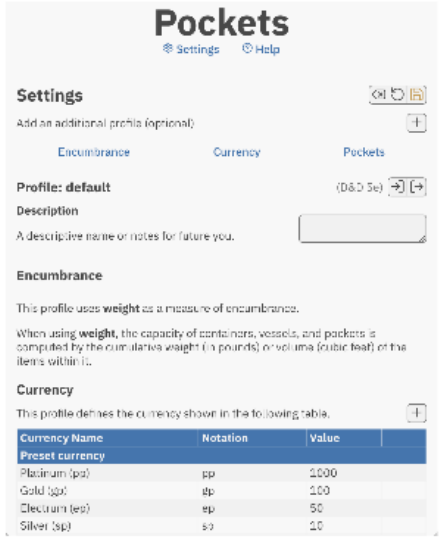
Pockets: The plan



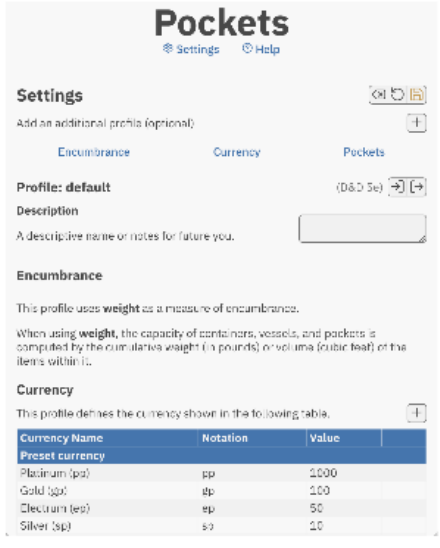
Pockets: The plan



Pockets: The plan



Pockets: The plan



REST and REST...

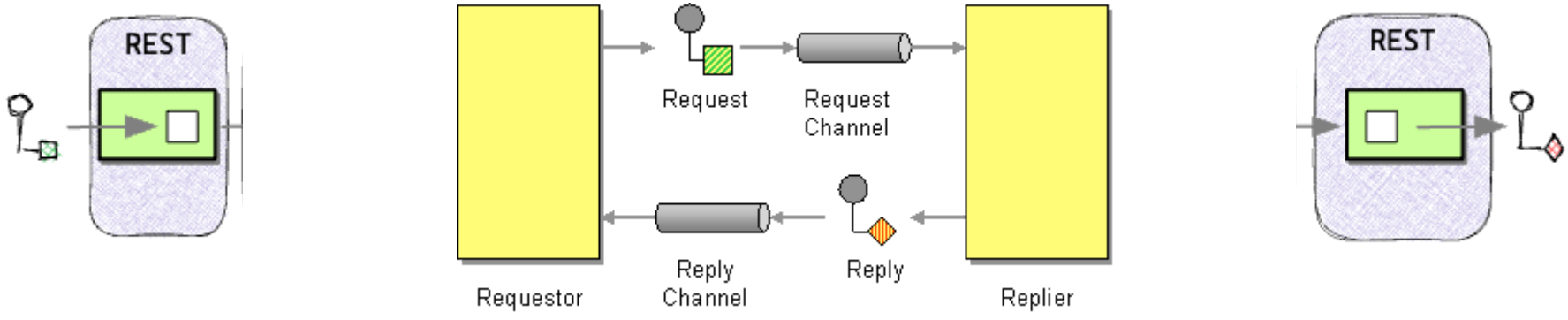
```
@Path("/pockets")
public class TheoreticalPockets {
    @GET
    public Response getProfiles() {
       // fetch some stuff
    }

    @GET
    @Path("{profile}")
    public Response getProfile(@PathParam("profile") String id) {
       // fetch specific stuff
    }

    @POST
    @Path("{profile}")
    public Response updateProfile(@PathParam("profile") String id, ...) {
        // do important things
    }
}
```

REST and REST...

```
    restConfiguration()
            .bindingMode(RestBindingMode.json)
            .componentProperty(key:"lazyStartProducer", value:"true")
            .dataFormatProperty(key:"autoDiscoverObjectMapper", value:"true");

    // -- Fetch presets

    rest(path:"/preset")
            .get(uri:"/{preset}")
            .to(uri:"direct:getPreset");

    // -- Get/Put config data (all profiles)

    rest(path:"/config")
            .get(uri:"current")
```

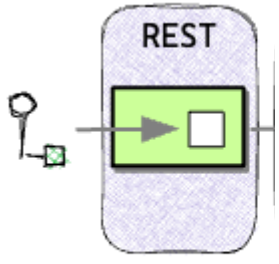
REST *component* adds support for REST semantics, including a DSL.

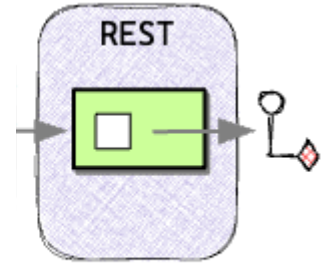
Flow of data with Camel Route: REST




```
1       rest("/pockets")
2               .get()
3               .to("direct:getProfiles")
4
5               .get("/{profile}")
6               .to("direct:getProfile")
7
8               .post("/{profile}")
9               .to("direct:validateAndModifyPockets"); // <--- HERE
```

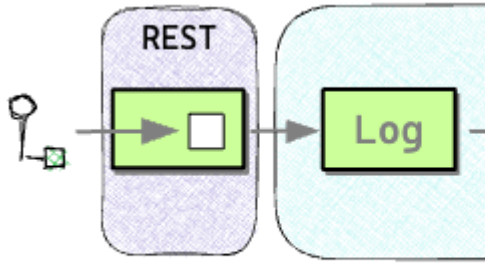
Flow of data with Camel Route: REST

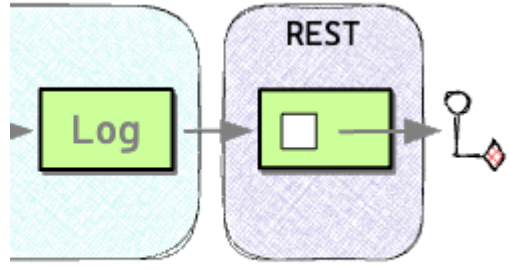





```
from(uri:"direct:validateAndModifyPockets")
    .autoStartup(autoStartup:false)
    .onException(exceptionType:InvalidPocketState.class)
        .handled(handled:true).process(new HandleInvalidPocketState())
    .end()
```

Flow of data with Camel Route: REST

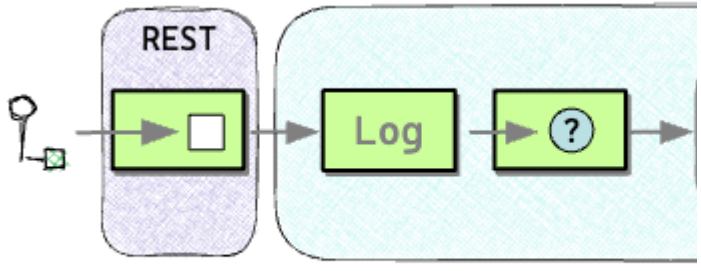


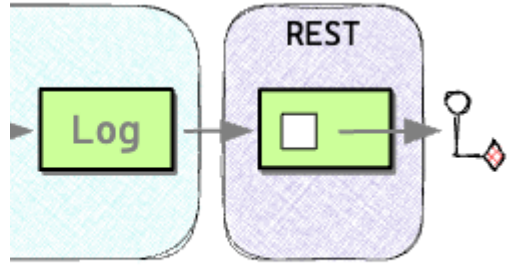



```
from(uri:"direct:validateAndModifyPockets")
    .autoStartup(autoStartup:false)
    .onException(exceptionType:InvalidPocketState.class)
        .handled(handled:true).process(new HandleInvalidPocketState())
    .end()

    .to(uri:"log:dev.ebullient.pockets?showAll=true&multiline=true&level=DEBUG")
```

```
    .to(uri:"log:dev.ebullient.pockets?showAll=true&multiline=true&level=DEBUG");
```

Flow of data with Camel Route: REST

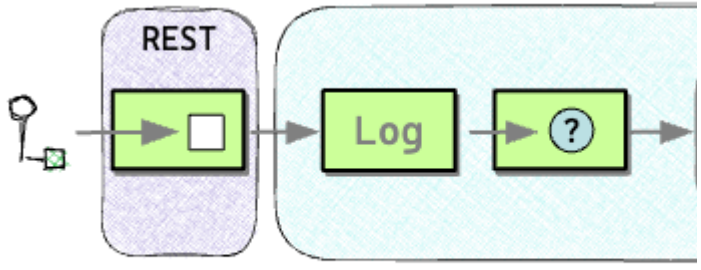


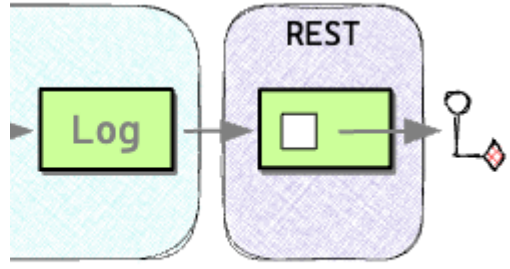



```
from(uri:"direct:validateAndModifyPockets")
    .autoStartup(autoStartup:false)
    .onException(exceptionType:InvalidPocketState.class)
        .handled(handled:true).process(new HandleInvalidPocketState())
    .end()

    .to(uri:"log:dev.ebullient.pockets?showAll=true&multiline=true&level=DEBUG")

    .process((e) -> webRouteHandler.validateProfile(e))
```

```
    .to(uri:"log:dev.ebullient.pockets?showAll=true&multiline=true&level=DEBUG");
```

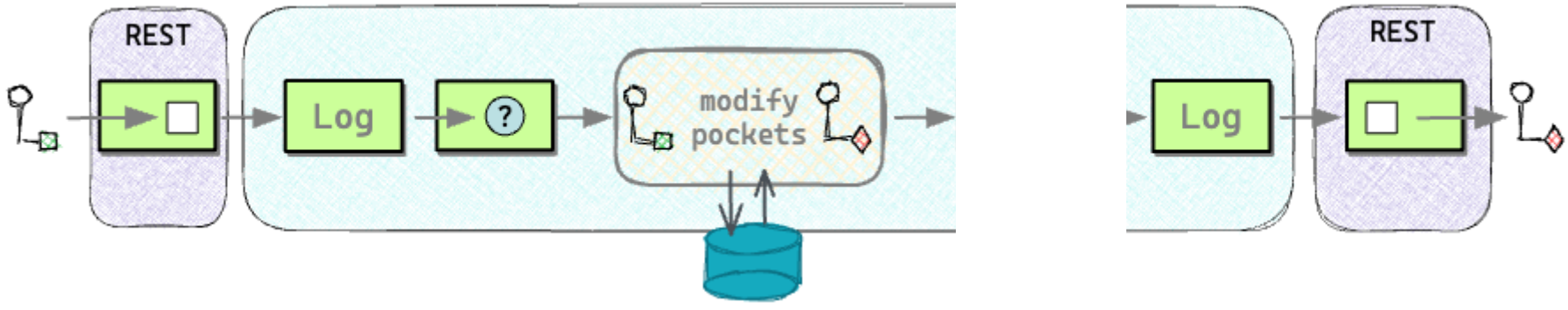
Flow of data with Camel Route: REST





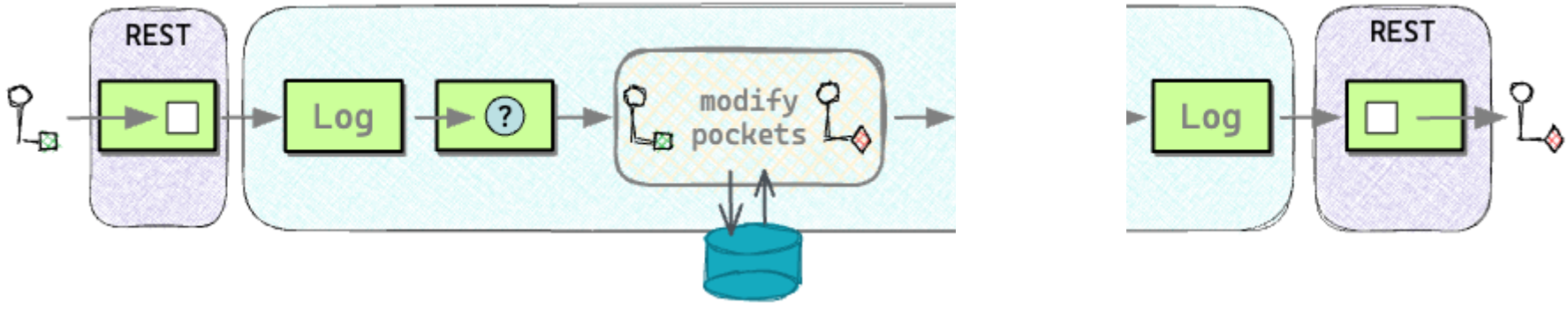

```
1       @Transactional
2       public void validateProfile(Exchange exchange) {
3           final String profileKey = "profile";
4
5           Message message = exchange.getMessage();
6           String profile = (String) message.getHeader(profileKey);
7
8           // Update header with full name, may throw
9           profile = ProfileContext.validateProfile(profile);
10
11          message.setHeader(profileKey, profile);
12      }
```

Flow of data with Camel Route: REST



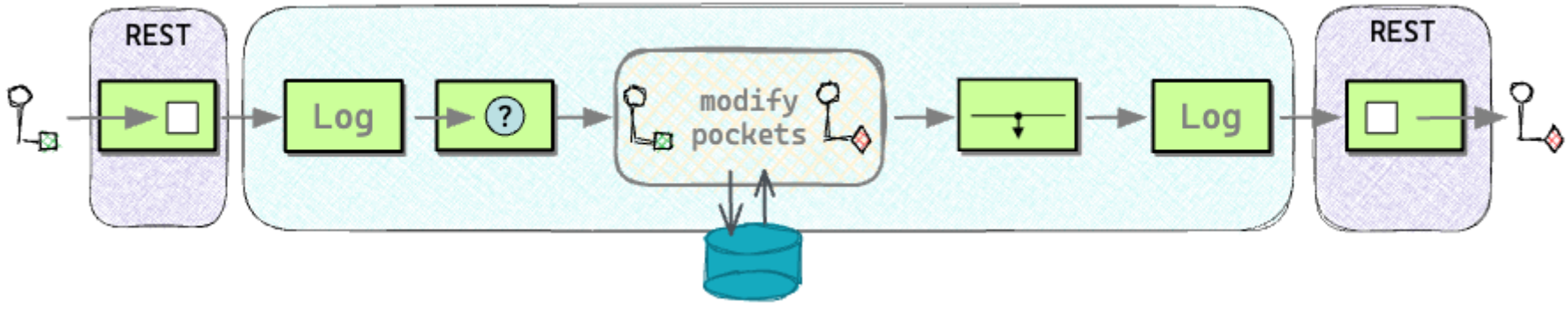

```
from(uri:"direct:validateAndModifyPockets")
    .autoStartup(autoStartup:false)
    .onException(exceptionType:InvalidPocketState.class)
        .handled(handled:true).process(new HandleInvalidPocketState())
    .end()

    .to(uri:"log:dev.ebullient.pockets?showAll=true&multiline=true&level=DEBUG")

    .process((e) -> webRouteHandler.validateProfile(e))

    .bean(webRouteHandler, method:"doModification(${body}, ${header.profile})")


    .to(uri:"log:dev.ebullient.pockets?showAll=true&multiline=true&level=DEBUG");
```

Flow of data with Camel Route: REST




```
1  .bean(webRouteHandler, "doModification(`${body}, $`{header.profile})")
2
3  ~~~~~~
4
5      @Transactional
6      public ModificationResponse doModification(
7              ModificationRequest modificationRequest,
8              String profileName) {
9
10         profileContext.setActiveProfile(profileName);
11
12         return modifyPockets.modifyPockets(modificationRequest);
13     }
```

Flow of data with Camel Route: REST




```
from(uri:"direct:validateAndModifyPockets")
    .autoStartup(autoStartup:false)
    .onException(exceptionType:InvalidPocketState.class)
        .handled(handled:true).process(new HandleInvalidPocketState())
    .end()

    .to(uri:"log:dev.ebullient.pockets?showAll=true&multiline=true&level=DEBUG")

    .process((e) -> webRouteHandler.validateProfile(e))

    .bean(webRouteHandler, method:"doModification(${body}, ${header.profile})")

    .wireTap(uri:"direct:emitRoute")

    .to(uri:"log:dev.ebullient.pockets?showAll=true&multiline=true&level=DEBUG");
```

Flow of data with Camel Route: CLI

```
1  @Override
2  @ActivateRequestContext
3  @Transactional
4  public final Integer call() {
5      try {
6          // Validate specified profile
7          String activeProfile = activeProfileMixin.findActiveProfileId();
8
9          if (activeProfile == null) {
10             return ExitCode.USAGE;
11         }
12
13         profileContext.setActiveProfile(activeProfile);
14         if (showTypes) {
15             showTypes();
16             return ExitCode.OK;
17         }
18
19         return delegateCall(); //  delegate to sub-command
20     } catch (...) {
21         ...
22     }
23 }
```

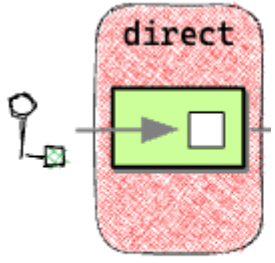
Flow of data with Camel Route: CLI

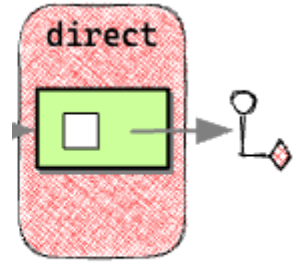
```
1       @Override
2       public Integer delegateCall() {
3
4           Modification modification = createModification();
5
6           ModificationRequest req = createModificationRequest(
7                   "Create " + modification.itemDetails.name)
8                           .add(modification);
9
10          // Make the modification
11          ModificationResponse response = modifyPocketRoute.modifyPockets(req);
12
13          response.changes.stream().filter(c -> c.type == PostingType.CREATE)
14                  .forEach(c -> {
15                      ...
16                  });
17
18          return ExitCode.OK;
19      }
```

Flow of data with Camel Route: CLI

```
1  // Make the modification
2  ModificationResponse response = modifyPocketRoute.modifyPockets(req);
3
4  ~~~~
5
6  public interface CliModifyPocket {
7      ModificationResponse modifyPockets(ModificationRequest request);
8  }
9
10 @Produce("direct:modifyPockets")
11 CliModifyPocket modifyPocketRoute;
```

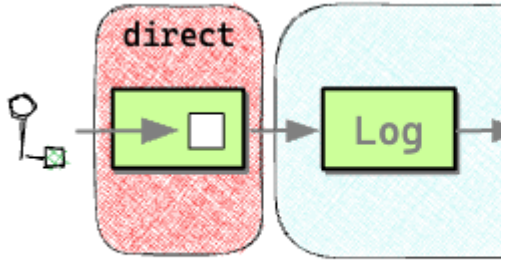
Flow of data with Camel Route: CLI

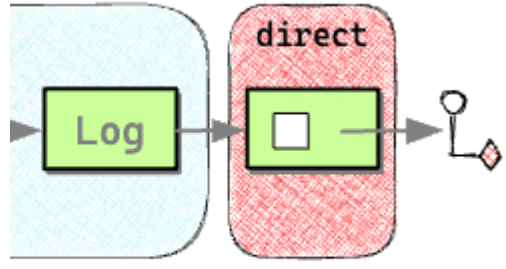





```
1 from("direct:modifyPockets")
2     .errorHandler(noErrorHandler())
```

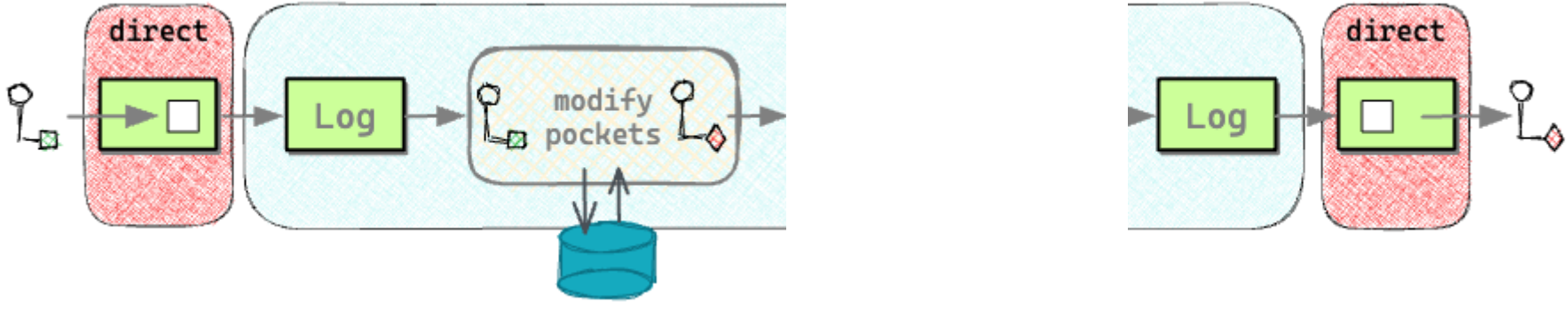
Flow of data with Camel Route: CLI





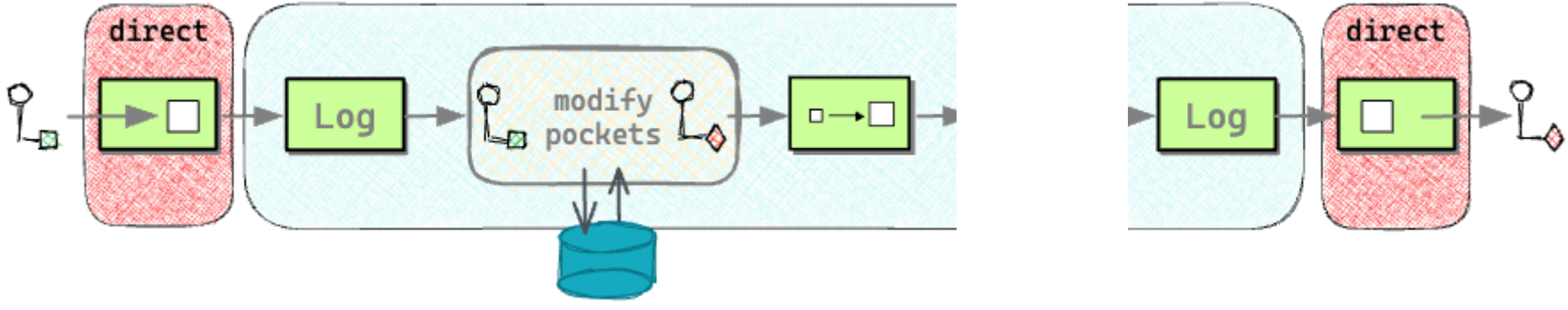

```
1  from("direct:modifyPockets")
2      .errorHandler(noErrorHandler())
3
4      .to("log:dev.ebullient.pockets?level=DEBUG")
5
12     .to("log:dev.ebullient.pockets?level=DEBUG");
```

Flow of data with Camel Route: CLI



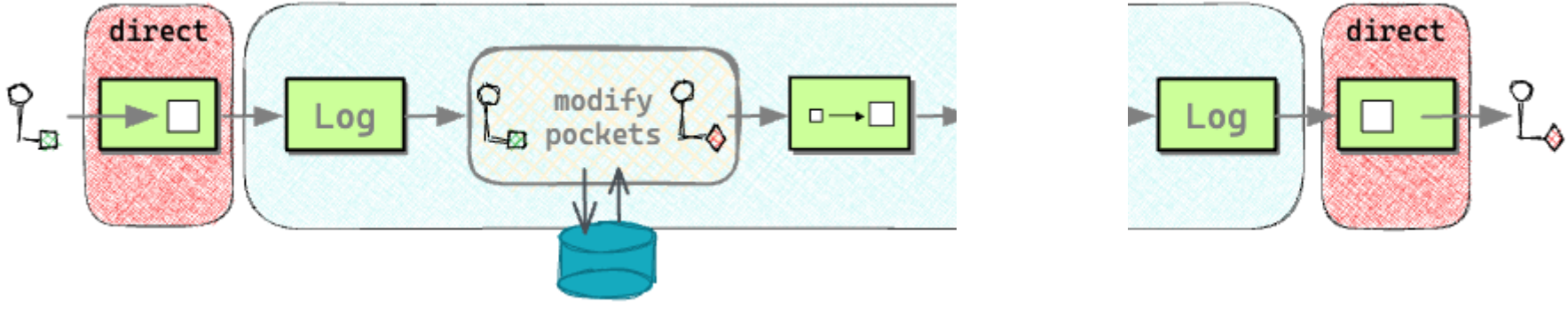

```
1  from("direct:modifyPockets")
2      .errorHandler(noErrorHandler())
3
4      .to("log:dev.ebullient.pockets?level=DEBUG")
5
6      .bean(modifyPockets, "modifyPockets(${body})")
7
12     .to("log:dev.ebullient.pockets?level=DEBUG");
```

Flow of data with Camel Route: CLI



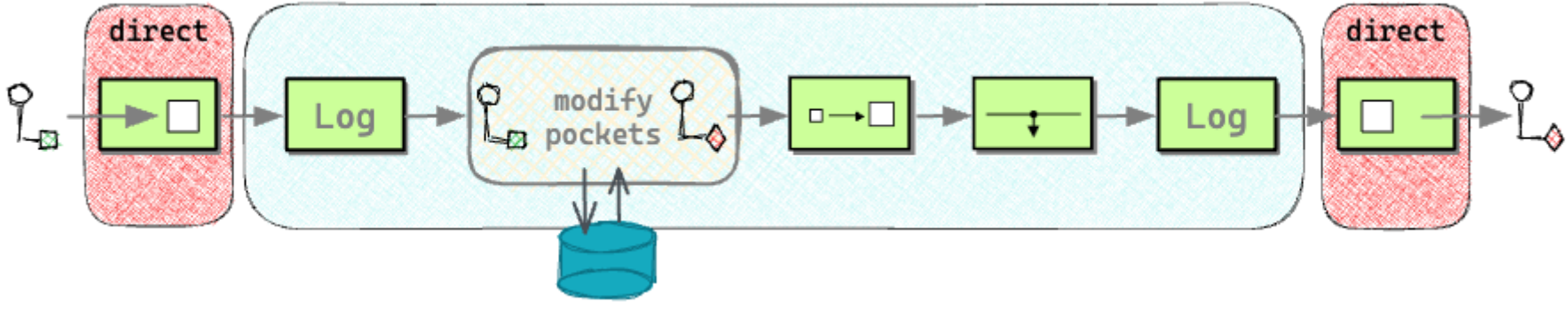

```
1  from("direct:modifyPockets")
2      .errorHandler(noErrorHandler())
3
4      .to("log:dev.ebullient.pockets?level=DEBUG")
5
6      .bean(modifyPockets, "modifyPockets(${body})")
7
8      .process((e) -> emitHandler.setProfile(e))
9


12     .to("log:dev.ebullient.pockets?level=DEBUG");
```

Flow of data with Camel Route: CLI




```
1 public void setProfile(Exchange e) {
2     Message msg = e.getMessage();
3
4     ModificationResponse resp = msg.getBody(ModificationResponse.class);
5
6     msg.setHeader("profile", resp.profileName);
7 }
```

Flow of data with Camel Route: CLI




```
1  from("direct:modifyPockets")
2      .errorHandler(noErrorHandler())
3
4      .to("log:dev.ebullient.pockets?level=DEBUG")
5
6      .bean(modifyPockets, "modifyPockets(${body})")
7
8      .process((e) -> emitHandler.setProfile(e))
9
10     .wireTap("direct:emitRoute")
11
12     .to("log:dev.ebullient.pockets?level=DEBUG");
```

Common path: WireTap

What does this do?

```
.wireTap("direct:emitRoute")
```

Common path: WireTap

What does this do?

```
.wireTap("direct:emitRoute")
```

It kicks off a new route to transform content

Common path: WireTap

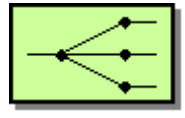
What does this do?

```
.wireTap("direct:emitRoute")
```

It kicks off a new route to transform content

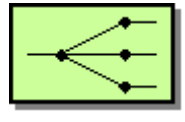
```
1 from("direct:emitRoute")
2     .to("log:dev.ebullient.pockets?level=DEBUG")
3     .bean(emitHandler, "emitRoute")
4     .recipientList(header("routes"))
5             .parallelProcessing();
```

Common path: Recipient List



```
@SuppressWarnings("unused")
public void emitRoute(Exchange exchange) {
    Message msg = exchange.getMessage();
    String profileName = msg.getHeader("profile", "default", String.class);

    List<String> emitters = new ArrayList<>();
    if (localPocketsConfig.jsonEventLogEnabled(profileName)) {
        EventLogConfiguration eventLogConfig =
                localPocketsConfig.eventLogConfig(profileName);
        msg.setHeader("jsonEventLogDirectory",
                eventLogConfig.getDirectory().toString());
        emitters.add("direct:jsonEmitter");
    }
    if (localPocketsConfig.markdownEnabled(profileName)) {
        emitters.add("direct:markdownEmitter");
    }
    msg.setHeader("routes", String.join(",", emitters));
}
```

Common path: Recipient List



```
1 from("direct:emitRoute")
2     .to("log:dev.ebullient.pockets?level=DEBUG")
3     .bean(emitHandler, "emitRoute")
4     .recipientList(header("routes"))
5             .parallelProcessing();
```

Common path: `direct:jsonEmitter`

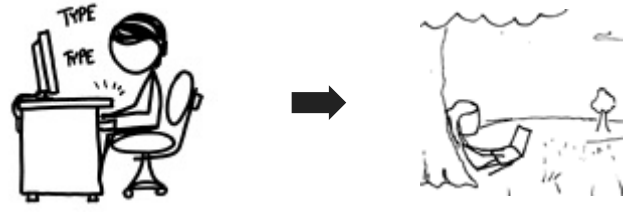
```
1  from("direct:jsonEmitter")
2      .onException(InvalidPocketState.class)
3          .handled(true)
4          .process(new HandleInvalidPocketState())
5      .end()
6
7      .setHeader(FileConstants.FILE_NAME,
8          simple("${header.profile}.events.jsonl"))
9
10     .setBody((e) -> Transform.toJsonString(e.getMessage().getBody()))
11
12     .toD("file:${header.jsonEventLogDirectory}?"
13             + "fileExist=Append&appendChars=\n&allowNullBody=true")
14
15     .to("log:dev.ebullient.pockets?level=DEBUG");
```

Common path: Markdown emitter

```
from(uri:"direct:markdownEmitter")
    .onException(exceptionType:InvalidPocketState.class)
        .handled(handled:true)
        .process(new HandleInvalidPocketState())
    .end()

    .bean(emitHandler, method:"emitMarkdownFiles(${body}, ${header.profile})")

    .to(uri:"log:dev.ebullient.pockets?level=DEBUG");
```

Common path: Qute templates

```
1  from("direct:markdownFile")
2      .onException(InvalidPocketState.class)
3          .handled(true)
4          .process(new EmitRouteHandler.HandleInvalidPocketState())
5      .end()
6
7      .to("qute:dummy?allowTemplateFromHeader=true")
8
9      .toD("file:${header.markdownDirectory}?" +
10         "fileExist=Override&allowNullBody=true")
11
12     .to("log:dev.ebullient.pockets?level=DEBUG");
```


TYPE
TYPE

Summary

- Apache Camel can help fill in the gaps
- Use it alongside your favorite messaging solution (Kafka or traditional message brokers)

Summary

- Apache Camel can help fill in the gaps
- It applies to patterns far beyond REST and CRUD
- Use it alongside your favorite messaging solution (Kafka or traditional message brokers)

Summary

- Apache Camel can help fill in the gaps
- It applies to patterns far beyond REST and CRUD
- Use it alongside your favorite messaging solution (Kafka or traditional message brokers)
- Use it with your favorite Java framework

Summary

- Apache Camel can help fill in the gaps
- It applies to patterns far beyond REST and CRUD
- Use it alongside your favorite messaging solution (Kafka or traditional message brokers)
- Use it with your favorite Java framework
- Use it with JBang for script nonsense

Summary

- Apache Camel can help fill in the gaps
- It applies to patterns far beyond REST and CRUD
- Use it alongside your favorite messaging solution (Kafka or traditional message brokers)
- Use it with your favorite Java framework
- Use it with JBang for script nonsense
- Use it with Kubernetes to make the best glue (Camel-K, Kamelets)

~fin~