

Quarkus Observability: Dev Services

Aleš Justin, OpenBlend / Red Hat

Lassie coming back home #3 :-)

Dev service?

- . An external service
 - . e.g. database, Kafka broker, ...
 - . BUT used at development / test NOT prod
- . Usually TestContainer + Docker image
 - . Extending GenericContainer
 - . Easy to get a "clean slate"
- . Quarkus already has a bunch of them
- . But let's make this even easier ... (from our past experience)

The problem?

- . Background
 - . Observability is (usually) not a single service, but a stack of related services
 - . There is always something you can squeeze into that stack
 - . Easy / flexible service selection from that stack
 - . Add your own service
 - . e.g. run any (Docker) image

Enter extensions 101

- Entry into Quarkus' "internals"
 - Eating your own dog food aka features
- Pre-processed "processors" with BuildItems
 - Ordered BuildItems
 - Produces vs Consumes
 - Multi vs single BuildItem
- Deployment vs Runtime modules
 - Only access to Runtime classes
- Recorders
 - Prepare / "record" bytecode steps
 - Access to config classes
 - Performance gain when metadata is involved
- List of capabilities - conditional items, etc

Dev Resource(s)

- . A bit more fine-grained "service"
 - . Otherwise we would be copy-pasting code all the time
- . Simple "contract" / signature
 - . Extends `QuarkusTestResourceLifecycleManager` (*)
 - . Allows for container re-usage (TODO: make this better)
 - . Container abstraction — no need for `TestContainers`
- . Comes in 3 "flavors"

Dev Resources via Dev Service(s)

- . Done in the same way as other Quarkus Dev Services
 - . But handling fine-grained Dev Resource(s)
 - . Via META-INF/services ... of course ;-)
 - . Smart(er) log - see actual config output
- . Hot re-deploy handling
 - . No config change = no new service
- . Re-use existing (Docker) containers
 - . Check for labels
 - . A bit of port fiddling ...

Dev Resources via ConfigSource

- “Lightweight” version of Dev Resources usage
 - No re-usage, no config check (and ref holding) on re-deploy, ...
 - Make sure to disable auto Dev Services and enable Dev Resources
- Use custom ConfigSource to start all found DevResources
 - Old-school META-INF/services mechanism
 - First call to “getPropertyNames()” or “getValue(name)”
- Discovery
 - Initially
 - As usual META-INF/services
 - But we’re in an extension?!
 - Now
 - RuntimeConfigurationSourceBuildItem
- Handle shutdown → ShutdownContext::addLastShutdownTask

Dev Resources via QuarkusTestResource

- Programmable way to pick Dev Resource(s)
 - Either
 - Disable auto(magical) lookup
 - Leave extension from the classpath

```
@QuarkusTest
@QuarkusTestResource(value = LgtmResource.class,
                    restrictToAnnotatedClass = true)
@TestProfile(QuarkusTestResourceTestProfile.class)
@DisabledOnOs(OS.WINDOWS)
public class LgtmLifecycleTest extends LgtmTestBase {
```

Config ...

- Dev Resources

```
quarkus.observability.dev-resources=true  
quarkus.observability.enabled=false
```

- QuarkusTestResourceLifecycleManager

```
quarkus.observability.dev-resources=false  
quarkus.observability.enabled=false
```

New Dev Resource?

```
public class LgtmResource extends ContainerResource<LgtmContainer, LgtmConfig> {

    @Override
    public LgtmConfig config(ModulesConfiguration configuration) {
        return configuration.lgtm();
    }

    @Override
    public Container<LgtmConfig> container(LgtmConfig config, ModulesConfiguration root) {
        return set(new LgtmContainer(config));
    }

    @Override
    public Map<String, String> config(int privatePort, String host, int publicPort) {
        switch (privatePort) {
            case ContainerConstants.GRAFANA_PORT:
                return Map.of("quarkus.grafana.url", String.format("%s:%s", host, publicPort));
            case ContainerConstants.OTEL_GRPC_EXPORTER_PORT:
            case ContainerConstants.OTEL_HTTP_EXPORTER_PORT:
                return Map.of("quarkus.otel-collector.url", String.format("%s:%s", host, publicPort));
        }
        return Map.of();
    }

    @Override
    protected LgtmContainer defaultContainer() {
        return new LgtmContainer();
    }

    @Override
    public Map<String, String> doStart() {
        String host = container.getHost();
        return Map.of(
            "quarkus.grafana.url", String.format("%s:%s", host, container.getGrafanaPort()),
            "quarkus.otel-collector.url", String.format("%s:%s", host, container.getOtlpPort()));
    }

    @Override
    public int order() {
        return DevResourceLifecycleManager.GRAFANA;
    }
}
```

Current status

- (Grafana) LGTM already available in 3.11
- And the whole Dev Resource implementation / feature
- More Dev Resources in new PR(s)
 - VictoriaMetrics(Agent)
 - Jaeger
 - OTEL Collector
 - Grafana (*)
 - Prometheus (*)
 - ...