Micronaut 'dev mode' Run Support

DRAFT

Micronaut applications can be run in **automatic restart** (Continuous Run) mode. In this mode, a change to a project source will trigger

- compilation
- packaging
- application restart

This run mode is **not suitable** for debugging, as the developer routinely modifies the underlying source, and (seemingly random) restarts would break the debugging context. There still needs to be a "traditional" run mode for work scenarios, where (random) restarts implied by source changes could be harmful, e.g. when the worked-on app maintains some connections to other services, or keeps some context.

The actual implementation should be **always delegated** to the underlying build/launch system.

- **gradle** supports **continous build mode** with **-t** parameter to the gradle launcher (read more about the mode)
- **maven** is extended by **micronaut** plugin and goal **mn:run** to do the same

Run in Dev mode, Debug normally

There is a UI-less way to enable support for mn:run when executing *Run (Single)* action and leaving behavior of *Debug (Single)* untouched. If we provide good enough extension API, then the Micronaut support module may remap the *Run* from **exec:exec** to **mn:run** for all `pom.xml` file that contain `microrna ut-maven-plugin` . Both Maven and Gradle project support action configuration mapping, so implementation in both Gradle and Maven will just provide a different **action mapping** for the existing project action. Gradle supports the **--continuous** flag from version 4.0.x (7 / 2017), so it seems pretty safe to add that action **unconditionally** to the project menu.

Should there be a necessity to support multiple types of Run or Debug, then read on...

Using project Configurations

The "Continuous run" is essentially a flavour, or variant of "Run" actions. With Gradle, "dev mode" can be applied to **run and test**. RunSingle

With Micronaut Maven plugin, however, only **Run** actions are supported. NetBeans **Maven** plugin support **Configurations**

- can redefine any **project action**
- can enable **a profile**

The downside is that project action definitions are completely independent: a change in main class, VM parameters or parameters must be copied over to all configuration(s) manually where the action is redefined: but that's not indicated at all to the user. **Gradle does not support configurations at all**. While Gradle plugin supports **action mapping**, similar to our Maven module, it does not contain the 2nd axis (configurations, profiles). If added, would be unique to the IDE (but profile-based action mapping in Maven is as well), without connection to the gradle build.

Continuous run acts as like a modifier on (certain) actions; it's a question how we handle a situation when there are **more such modifiers** that could be combined, i.e.

- continuous run
- logging
- database connections

as each single is (now) represented by a Configuration - which do not combine or merge.

Separate Project Action(s)

A new project action **Continuous Run** can be defined to handle this special action. Both Maven and Gradle project support action configuration mapping, so implementation in both Gradle and Maven will just provide an **action mapping** for the new project action. The project action itself should be defined in

- **Gradle Projects** module: as **-t** is a Gradle feature, this will enable its use in other Gradle projects, not just Micronaut.
- **Micronaut** module (for Maven): Maven does not support it itself, but Micronaut's plugin does. Possibly, to reduce dependencies, **micronaut.maven** bridge module could be created.

This Project Action can be then invoked programmatically, apart from the project UI itself, by a LSP client, too.

❓ Since the action can be seen as a **replacement** for traditional Run (depending on project characteristics and user's preferred workflow), it should be configurable as the **default Run action**, responding for

- ActionProvider.COMMAND_RUN,
- ActionProvider.COMMAND_RUN_SINGLE.

⚠️ Need to define **how** an (abstract) "Run" action can be remapped to a different action ("run-continuous") defined in **nbactions.xml**.

Definition for Maven

```
    <action>
        <actionName>run.continuous</actionName>
        <packagings>
            <packaging>jar</packaging>
        </packagings>
<!--
                Maybe not necessary to add, if the MavenActionsProvider itself are registered specifically per-
plugin

        <activation>
            <plugin>io.micronaut.build:micronaut-maven-plugin</plugin>
        </activation>
-->
        <goals>
            <goal>process-classes</goal>
            <goal>io.micronaut.build:micronaut-maven-plugin:2.0.0:run</goal>
        </goals>
        <properties>
            <mn.jvmArgs>${exec.vmArgs} -classpath %classpath</mn.jvmArgs>
            <mn.appArgs>${exec.appArgs</mn.appArgs>
            <exec.mainClass>${packageClassName}</exec.mainClass>
            <exec.executable>java</exec.executable>
        </properties>
    </action>
```

Display Action in the IDE

Neither Gradle or Maven support adding **technology-specific** actions in the project UI. Even if the action is defined in **nbactions.xml**, it will not appear anywhere in the popup menu, except for **custom actions** (having CUSTOM- prefix in their action name; maven only). In this case, I would like to display the **Continuous Run** action alongside the **Run** and **Debug**.

- Gradle supports the **--continuous** flag from version 4.0.x (7 / 2017), so it seems pretty safe to add that action **unconditionally** to the project menu. There are some **limitations** documented, the action **could** warn the user for the 1st time the action is used, if it encounters such an environment.
- The action should not be present for **other than micronaut** projects (those which contain MN plugin) in Maven.

⚠️ **Need to define a** representation of such configuration in the UI. An alternative would be a menu item that

- executes on click,
- can display a submenu which contains
    - Run once
    - Run continuous
- the new state would be remembered

Adding technology-specific actions

Common actions for a project type are placed in **Projects/<project-type-id>/Actions**. These actions are displayed in the Project's context menu in the filesystem-defined order. I propose to define a **layer API** that would include also actions from **Projects/<project-type-id>/plugin-id/Actions** for all *plugins* participating on the project:

- plugins referenced in the **build.gradle**,
- plugins **configured** by **active profiles** of the maven project

➕ Allowing to extend the main project menu may lead to its explosion with many added actions for each technology. A standard **grouping action** should be created in the OpenAPI and documented, so a technology may eventually add its action into a subgroup. This **will not be part** of the initial implementation, but could be added later. Note that by default, the **Lookups.forPath()** collects the whole .../Actions subtree (traversing into subfolders).

Integration with VSCode

Launching run configurations

The **java8+** launch type should be enhanced with either

- continuousExecution : boolean | null, or
- **configuration : string | null**, that would select the desired configuration known by LSP server
- **action**: **string** | **null**, to select the (abstract) action
- **configurationParams**: generic Map to support possible future launch extensions

The user may edit / change the Run configuration or create an additional one that executes the application in the "Continuous run" mode. Blank **action** would enable the same logic, as it is done now. Unhandled actions would get the **FileObject** of the active file in its actionLookup - that would eventually enable us to allow more flexibility through DAP protocol.

Access to run configurations

Run configurations extracted from the project would be served over LSP to **DebugConfigurationProvider** that should report them from **provideDebugConfigurations**.

Project Actions - implementation

Action contributions

Action has to be contributed to the project based on **plugin**, This is supported with Gradle (but gradle has the action centralized in the core), but must be added to Maven.

RunJar support

Current "Run" operation relies on **RunJar** prerequisity checker and late-bound checker to step in, and supply necessary values for ${} variables referenced in the action config. The prerequisity checker checks for **specific action IDs** when activating. If Micronaut support defines **Continuous Run** as a different action ID, these checkers **will not** be used, and the action may become broken: for example application args, VM args etc as seen in the Project Properties dialog will not be processed, **StartupExtenders** will not be collected etc. Two (or more ?) options here:

- Micronaut plugin will **duplicate** the RunJar logic, or
- We export an API from Maven core "Run" support to **bind** that support to specific executions, so the logic could be reused
    - packaging (already done)
    - project action ID
    - active plugins
- We export an API from either Maven (but see Gradle notes below) or from Project support itself, to **categorize** actions. Something like **boolean isKindOf(abstractActionId, actionId)**, with an appropriate (declarative ?) SPI that Micronaut can use to declare **Continous Run** is kind of Run. This way, **runJar** support may check for **categories** instead of action Ids.

The logic implemented by Maven (which can be reused by Micronaut) is:

- Selection of the main class: if not defined, UI shows up, eventually recording user's choices in **nbactions.xml**
- Java platform selection
- StartupExtender VM parameters merge
- ExplicitProcessParameters processing

Micronaut Maven Plugin

While **exec:exec** goal honours system property **exec.executable**, that eventually specifies the desired **Java** executable to be used for application launch, Micronaut maven plugin **does not honour any such property** and relies on **toolchains** in maven. NetBeans **do not support** toolchains much, and our LSP clients also manage JDKs in a different way, not using maven toolchains. **Java platform selection will be broken** for mn:run goal, unless the **whole maven** runs on the target JDK.

The Micronaut plugin **does not honour** the following properties we need to transfer parameters from the IDE, or LSP client:

- mn.jvmArgs - just plugin configuration property is supported, not **system** property on commandline
- mn.appArg - just plugin configuration property is supported, not **system** property on commandline
- exec.executable - not supported **at all**, even as configuration propertyu
- exec.args - whole composed command line

The simplest solution is to **enhance the mn:run** goal to support some 'exec.executable'-like (i.e. **mn.executable**) property with a similar semantics.

Gradle implementation

The current parameter-passing implementation could work, **mostly**, except that it also specifically checks for action IDs. Since Gradle supports **Continuous run** natively, the JavaExecTokenProvider could also check for the new action ID. But if **action categorization** (as outlined for Maven) is invented, it would benefit from that - more actions (even user custom ones) could be categorized as "run-like". Other than that, gradle support should work acceptably.

Currently gradle support in NetBeans **lacks run configuration UI** - the user cannot specify application arguments, JVM arguments, env variables or main class.

Configurations - implementation

Maven

Configurations are already there. We need to make **ProjectConfigurationsProvider** instances plugin-aware, similar to the **ActionProvider** scenario. With **micronaut-maven-plugin** a "**Development mode**" configuration would be magically pre-defined with suitable defaults. Users can override. The idea is:

- make a configuration form **each profile** defined in effective POM

Gradle

Gradle itself does not support profiles, or any configurations. Maven profiles can be transposed into gradle using **conditional inclusion** of buildscript fragments.