TINIEST PAAS USER GUIDE

What is it

Tiniest PaaS allows **deploying and hosting** Java applications, providing users with a simplified (or even naïve) cloud-based 'Platform As A Service' environment. In the spirit of *make JAR, not WAR*, all it needs from the user is an executable jar, so that it can call `java -jar yourApplication.jar` to get the application up and running.

It means that:

- Tiniest Paas is not a Servlet Container, users are free to use any technology stack that can run on a JVM
- hosted application must provide a regular `public static void main(String[])` entry point, and a corresponding manifest entry, as per JVM specification
- hosted application must be packaged as a *fatjar*, with all its dependencies bundled inside
- if hosted application wants to serve network traffic, it must start an embedded server on a particular port itself (e.g. embedded Tomcat to handle http requests)

Provided Desktop Client is just a thin UI layer, connecting to a specified Tiniest PaaS server instance. It obviously does NOT run the hosted/deployed applications on user's machine, but rather forwarding all user operations to the PaaS server via HTTP/REST API calls.

What problem does it solve

If you developed a (web)app and would like to test it in a non-local environment, or just have it hosted somewhere, you usually need to:

- set up a Personal Project in EPAM cloud
- configure OS on the provided VM
- install Java on the VM
- install Tomcat on the VM
- deploy your app
- know how to at least reach logfiles of your app and Tomcat for basic monitoring and troubleshooting

If you are interested in developing your app rather than provisioning the environment and resources for it, Tiniest PaaS is for you.

What is it NOT

Tiniest PaaS is not a full-blown PaaS solution, so unlike industry-standard solutions:

- it does not offer full containerization (hosted apps share many resources of the host server, such as ports)
- it does not provide subdomains for the hosted apps
- it does not offer dynamic scaling (it will not spin up additional VMs if host's resources (memory, CPU) are exhausted)
- it does not offer any load balancing (in case you want to run multiple instances of your app)
- it is not a high-availability solution, so no SLA on uptime

All of these are subject to further development.

Therefore, at its core, Tiniest PaaS is just a sophisticated `ProcessBuilder` to execute your jar file on a Linux server, with a convenient UI to do it remotely ☺

Quick start

As noted previously, you can deploy and host **any executable *fatjar***. Easiest way to develop something working in no time is by using Spring Boot. Here is how you can have a deployable *'Hello world'* webapp in 5 minutes or less:

- Go to https://start.spring.io/
- Select 'Web' dependency
- Generate Project
- Import the project into your favorite IDE
- Add the following code to your `Application` class:

```
public static void main(String[] args) {
    SpringApplication.run(Application.class, args);
}

@RestController
static class Endpoint {

    @GetMapping("/")
    String helloWorld() {
        return "Hello world";
    }
}
```

- Build your project (e.g. `mvn package`), and voila, you are ready to deploy.

Disclaimer: default port that Spring Boot's embedded Tomcat will try to bind to is 8080, which is likely to be used by another application. You may want to change the default, e.g. by providing a command line parameter during deployment: `--server.port=xxxx`

After deployment, your application will be available at:

`http://[PaaS hostname]:xxxx/` (e.g. `http://ecsc00300356.epam.com:xxxx/`)

Provisioning

There are several services that Tiniest PaaS can provision for the hosted applications; they will be elaborated on later in this document.

At minimum, PaaS will intercept all output that a hosted application sends to `System.out` and `System.err`, keep a buffer of last 1000 lines of the output, and expose it to the user via Desktop Client.

Provisions – monitoring

In order to be subject to monitoring, the hosted app needs to expose certain REST endpoints, allowing Tiniest PaaS query for status, metrics, JMX, etc. Implementing such capabilities is a lot of work, so effectively this provision **only makes sense for Spring Boot apps built as** `spring-boot-admin-starter-client`.

The following dependency should be added to your Spring Boot app in order to be subject to monitoring:

```
<dependency>
    <groupId>de.codecentric</groupId>
    <artifactId>spring-boot-admin-starter-client</artifactId>
    <version>1.5.0</version>
</dependency>
```

Tiniest PaaS will start your app with necessary command line parameters, to enable automatic registration of your app for monitoring.

Monitoring UI can then be opened by clicking 'Open monitoring' button in Desktop Client (*Application details* section) or by pointing your browser to:
`http://[PaaS URL]/unrestricted/monitor/{appId}`

Monitoring logs.

Monitoring UI can observe and display your logs, if your Spring Boot app provides a proper endpoint. Easiest way to set it up (although not compliant with 12factor way due to file usage) is:

- Remove any custom logging configuration (e.g. `logback-spring.xml`), to use Spring Boot defaults
- Put the following in your `application.properties`:
    - `logging.file=myLogfile.log`

Monitoring UI will now have *Log* tab with your logs .

Disclaimer: Currently no security is applied to monitoring UI or your metrics endpoints, so anyone will be able to monitor your app.

Monitoring implementation: http://codecentric.github.io/spring-boot-admin/1.5.0/

Provisions - logging

Cloud-deployed apps must not assume anything about availability of storage for their logfiles. Please consider this section of the Twelve Factor App manifesto: https://12factor.net/logs

Apps hosted on Tiniest PaaS can opt for the following:

- Request the host to automatically publish System.***out*** and System.***err*** output to a shared ELK stack
- Request the host to provision monitoring. The monitoring solution offers access to logs via web interface (see previous section for details).
- Request the host to provide a Logstash http endpoint URL via command line parameter, so that the application's own logging system can have finer control over what and how gets published to the shared ELK stack.

Example of the last option is using a `LogstashTcpSocketAppender` in your application. In a Spring Boot app, the following piece of configuration can be placed in `logback-spring.xml` file in order to make use of the provided Logstash URL. **Please note that this is not a complete configuration of the appender, just a small excerpt.**

```
<springProperty name="logstash.destination" source="logstash.url"/>
<appender name="STASH" class="net.logstash.logback.appender.LogstashTcpSocketAppender">
    <destination>${logstash.destination}</destination>
        <!--rest of the configuration of the appender -->
```

Provisions – database

[description coming soon]

Provisions – local storage

[description coming soon]

Deployment automation

Once you deployed your application in Tiniest PaaS, it was assigned an `appId`. Subsequent redeployments of your application can be automated via a Maven plugin (`paas-maven-plugin`), using the **existing** `appId`.

This plugin will upload your build artifact (expectedly a *fatjar*) to the Tiniest PaaS server, stop your application, replace the previous jar with the new one, and restart the application. All other settings (provisions, command line parameters) will remain the same.

Configuration of the plugin is below. The plugin and its dependencies are hosted on EPAM's Artifactory server, so relevant sections pointing the corresponding repositories are required.

```
<build>
    <plugins>
        …
        <plugin>
            <groupId>com.epam.sandbox.paas</groupId>
            <artifactId>paas-maven-plugin</artifactId>
            <version>1.0-SNAPSHOT</version>
            <configuration>
                <paasServerUrl>http://ecsc00300356.epam.com:8080/paas</paasServerUrl>
                <appId>1</appId><!--application ID to redeploy new jar with-->
                <username>guest</username>
                <password>guest</password>
            </configuration>
        </plugin>
    </plugins>
</build>
…

<pluginRepositories>
    <pluginRepository>
        <snapshots>
            <enabled>true</enabled>
        </snapshots>
        <id>EPAM</id><!-- my .m2/settings.xml declares a server with this ID and with my EPAM credentials -->
        <url>https://artifactory.epam.com/artifactory/list/libs-snapshots-local</url>
    </pluginRepository>
</pluginRepositories>
```

```
<repositories>
    <repository>
        <snapshots>
            <enabled>true</enabled>
        </snapshots>
        <id>EPAM</id>
        <url>https://artifactory.epam.com/artifactory/list/libs-snapshots-local</url>
    </repository>
</repositories>
```