Mjolnirr: private PaaS as distributed computing evolution

Gleb Radchenko,
Dmitry Savchenko
South Ural State University, Russia
Problem definition

- Cloud computing enables resource providers to reduce support and integration costs, using elastic resource management.
- But public cloud platforms raise a security concern: data is stored and processed remotely.
- Private clouds are the only option for the company that want to provide computing resources inside the company.
  - But most of existing private cloud solutions provide IaaS level of clouds that often require complicated procedures for support and usage of resources.
Requirements

- **Mjolnirr platform** – solution for Java-based private PaaS systems deployment:
  - Provide an API to enable programmers to write new modules easily
  - Supports component-oriented loose-coupled system architecture
  - Provides automation of components distribution and deployment
  - Component containers can work not only on server hardware, but on end-user PCs
  - Provides integration with the UNICORE grid services
Mjolnirr platform Architecture

Diagram showing the architecture of the Mjolnirr platform, including components such as Web Browser, Proxy, Container, Component A, Message bus, Distributed filesystem access module, and Database access module.
Architecture: Proxy
Architecture: Container
Development: Components

• Two types of custom components:
  – Application provides the user interface, scripts and styles as static files that are packaged in a JAR archive as well as user requests processing logic
  – Module represents a single entity in the domain of the application. For example, the Person module will provide all the necessary methods for working with persons within the current application, but no more

• Developer:
  – Creates a component on the basis of provided API
  – Uploads the component to a Proxy, using the web-interface

• The component instances are deployed on containers automatically
Development: Component interface

@MjolnirrComponent(
    componentName = "calculator",
    instancesMinCount = 1,
    instancesMaxCount = 255,
    memoryVolume = 128)

public class Calculator extends AbstractApplication {
    private ComponentContext context;

    @MjolnirrMethod
    public String calculate(String expression)
    throws Exception {
        return Helper.calculate(expression);
    }

    @Override
    public void initialize(ComponentContext context) {
        this.context = context;
    }
}
Development: Component UI

• You can use jade as web-template engine and JavaScript to develop interactive UI

```javascript
function calc() {
    var inputField = $("#calculator-string");
    try {
        inputField.val(callRemoteMethodSync({
            method: "calculate",
            args: [ inputField.val() ]
        }));
    } catch (err) {
        bootbox.alert(err);
    }
}
```
Application execution
Performance evaluation

• 1 gigabyte of text data was divided on 100 parts and sent to all available worker components for processing.
• Each worker divide text on words and count a frequency of each unique word. Pieces of work were distributed automatically – each worker polled Message Bus to receive new task.
Experiments have shown that the platform is stable. Average execution time on 10 containers was 219 seconds. Thus, acceleration of parallel word frequency counter task was 5.3.
Further directions

• As a development of this project, we are planning to provide:
  – Messaging system evaluation;
  – Application-level migration support to provide system stability;
  – Resource monitoring for flexible load balancing;
  – Global component store to reduce the number of the duplicate applications;
  – Better component isolation to provide system security;
  – Integration modules for DBMS and distributed file-management systems.
Contacts

• Gleb Radchenko
  – gleb.radchenko@susu.ru

• Dmitry Savchenko
  – dmitry.savc@gmail.com

• All sources are available on BitBucket:
  – https://bitbucket.org/mjolnirr/mjolnirr/src
Container deployment

1.1 Request component names
1.2 Component names

[For each component]
2. Check component availability
2.1:

[No component in cache]
3.1: Request component
3.2: Component

4: Load the component