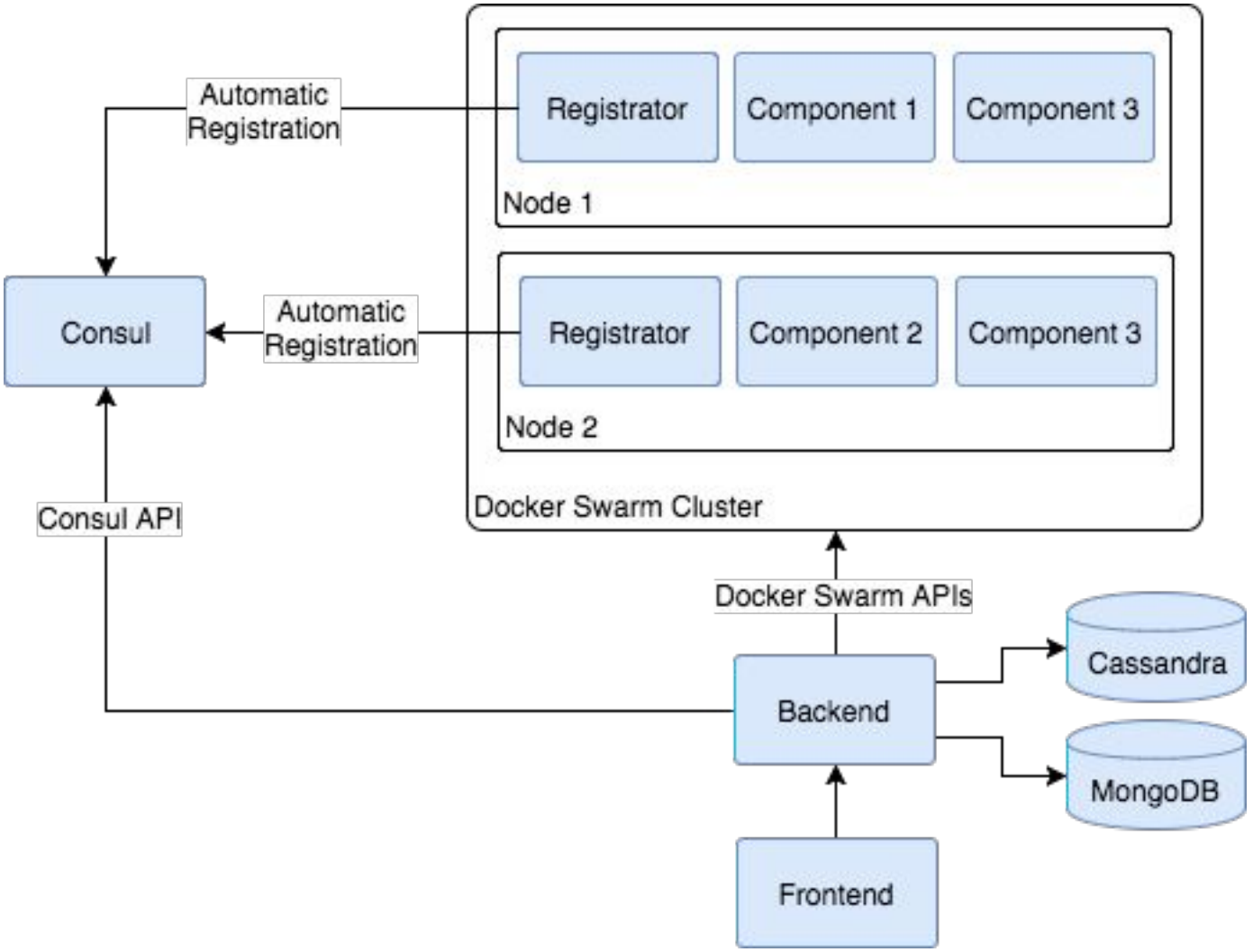
Consul Docker Management Dashboard

Group 4
Davide Pedranz, Francesco Segala,
Yuying Andrew Chen

Architecture



Architecture (cont)

- The monitored application runs in a Docker Swarm cluster.
- Each node of the cluster run an instance of Registrator that automatically register / deregister each container to Consul.
- Consul store the information about the registered components and their settings and provides service discovery.
- Our backend reads the state of the monitored application from Consul and uses the Docker Swarm APIs to run commands against the cluster.
- Our frontend uses the REST APIs of the backend to implement the dashboard.

Technology Stack

	Technology	Description
Frontend	Angular 4	Framework to build rich and responsive frontends.
	Angular Material	UI framework for Angular.
Backend	Play Framework	Web framework for Scala.
Storage	MongoDB	Store information about the users: users details, users credentials, permissions.
	Cassandra	Save store audit logs and the history (changes) in the monitored application [we have to verify if this is feasible].
	Consul	Store configuration parameters.
Other Software	Consul APIs	Read information about the registered components and change their configurations.
	Docker Swarm APIs	Perform operations on the cluster that hosts the monitored application: add new components, scale the number of instances, etc.

Data Structures

User information:

```
{
        "user_id": "some-uuid",
        "credentials": {
                "username": "xxx",
                "password": "yyy"
        },
        "details": {
                "email": "some@email.com",
                "name": "John"
        },
        "permissions": [
                "Read-status",
                "Change-settings",
                "Scale-containers"
        ]
}
```

Audit logs:

- uuid
- timestamp
- user_id
- action
- component
- additional_information

Cluster history:

- uuid
- timestamp
- component
- event
- additional_information

Plan of Implementation

1. Get familiar with Consul, automatic components registration and service discovery.
2. Select a useful subset of common operations performed on a Docker Swarm cluster and verify the available APIs.
3. Implement the frontend, with a mocked backend (this step will define the needed APIs to implement in the backend).
4. Implement the backend, using mocked in-memory storage.
5. Implement the database layer and connect it to the backend.
6. Deploy the software in a Docker Swarm cluster.
7. Test for bottlenecks, scalability and resilience to failures.