SISTEMAS DISTRIBUÍDOS

Aula 2

Trabalho prático

Deployment

AGENDA

Apresentação do trabalho

Deployment usando Docker

TRABALHO PRÁTICO

Serviço de indexação

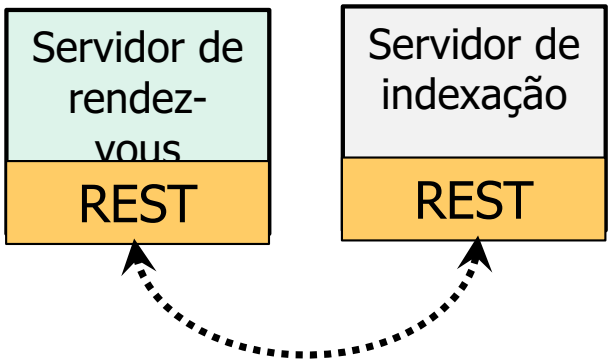
Permite pesquisar num conjunto de documentos

documento = (url, lista de keywords)

E.g. documento = tweet
url = URL para aceder ao tweet
lista de keywords = hashtags

NOTA: serviço de indexação não mantém documentos

TRABALHO PRÁTICO – FASE 1

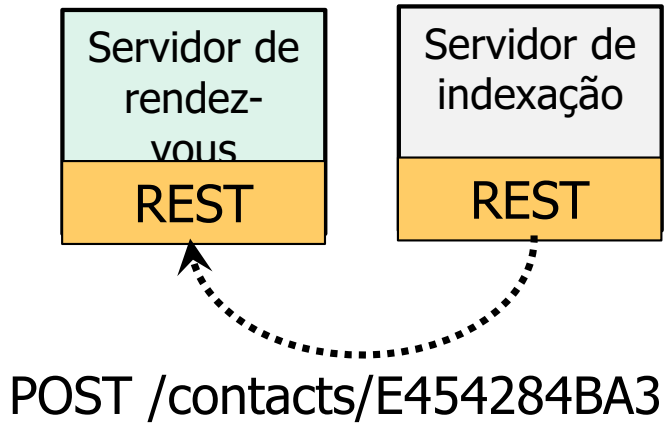


Servidor de rendez-vous
Informação sobre servidores de indexação

Interface do Servidor de Rendez-vous

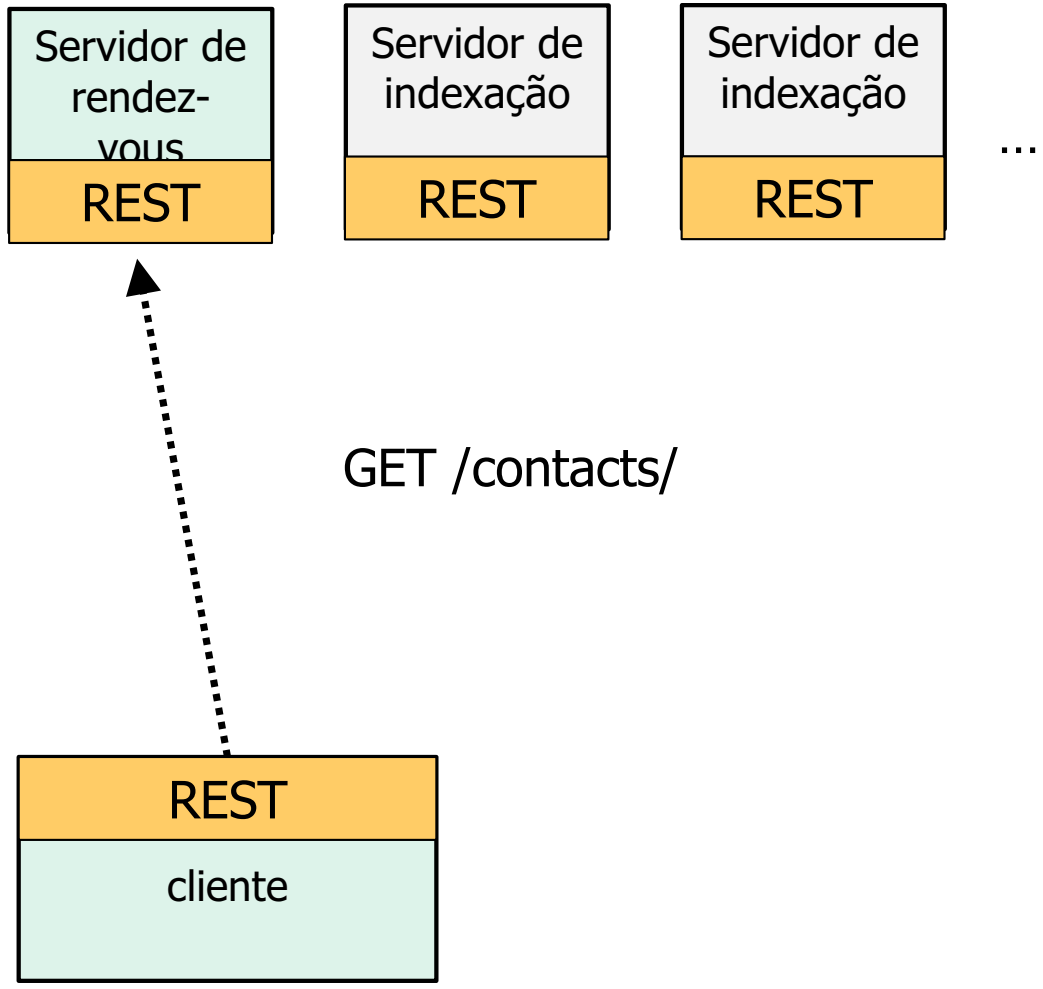
```
/**
 * Interface do servidor que mantem lista de servidores de indexacao.
 */
@Path("/contacts")
public interface RendezVousAPI {

    /**
     * Devolve array com a lista de servidores registados.
     */
    @GET
    @Produces(MediaType.APPLICATION_JSON)
    Endpoint[] endpoints();

    /**
     * Regista novo servidor.
     */
    @POST
    @Path("/{id}")
    @Consumes(MediaType.APPLICATION_JSON)
    void register(@PathParam("id") String id, Endpoint endpoint);

    /**
     * De-regista servidor, dado o seu id.
     */
    @DELETE
    @Path("/{id}")
    void unregister(@PathParam("id") String id);
}
```

Trabalho prático – fase 1




```
@POST
@Path("/{id}")
@Consumes(MediaType.APPLICATION_JSON)
void register(@PathParam("id") String id, Endpoint endpoint);
```

TRABALHO PRÁTICO – ADICIONAR INFORMAÇÃO SOBRE DOCUMENTO



Servidor de rendez-vous
Informação sobre servidores de indexação

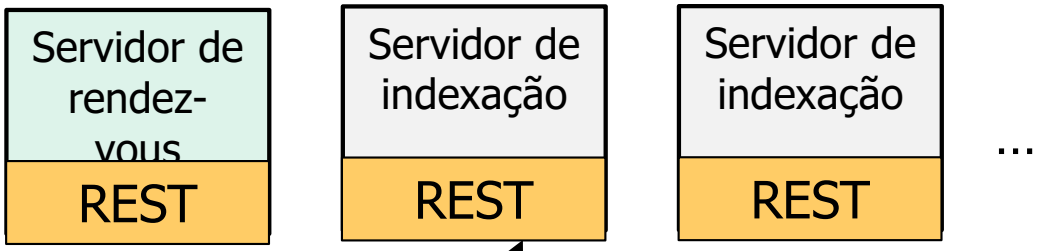
Servidor de indexação
Informação sobre documentos (URL, lista de keywords)

```
@GET
@Produces(MediaType.APPLICATION_JSON)
Endpoint[] endpoints();
```

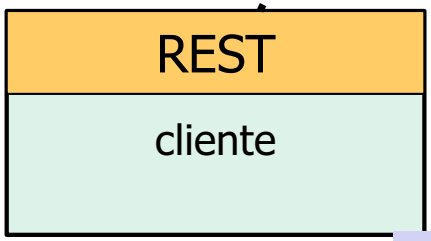
Interface do Servidor de Indexação

```
/**
 * Interface do servidor de inexacao.
 */
public interface IndexerService {

    /**
     * Devolve lista de URLs dadas as palavras chave keywords codificadas como:
     * palavra_1+palavra_2+...+palavra_n.
     */
    @GET
    @Path("/search")
    @Produces(MediaType.APPLICATION_JSON)
    List<String> search(@QueryParam("query") String keywords);

    /**
     * Adiciona informacao sobre documento doc com identificador id.
     */
    @POST
    @Path("/{id}")
    @Consumes(MediaType.APPLICATION_JSON)
    void add(@PathParam("id") String id, Document doc);

    /**
     * Remove informacao sobre documento com identificador id em todos os servidores.
     */
    @DELETE
    @Path("/{id}")
    void remove(@PathParam("id") String id);
}
```

TRABALHO PRÁTICO – ADICIONAR INFORMAÇÃO SOBRE DOCUMENTO

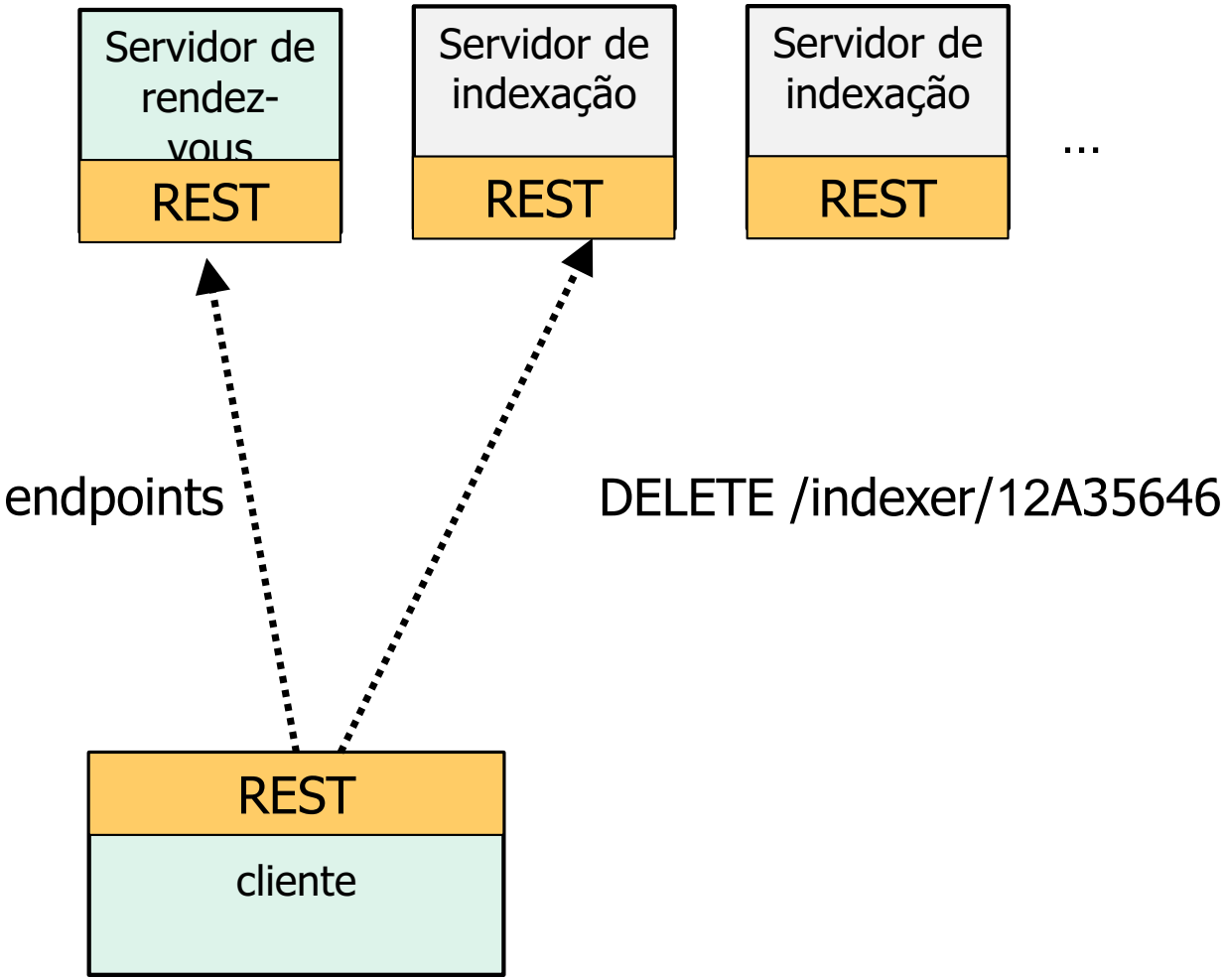


POST /indexer/12A35646



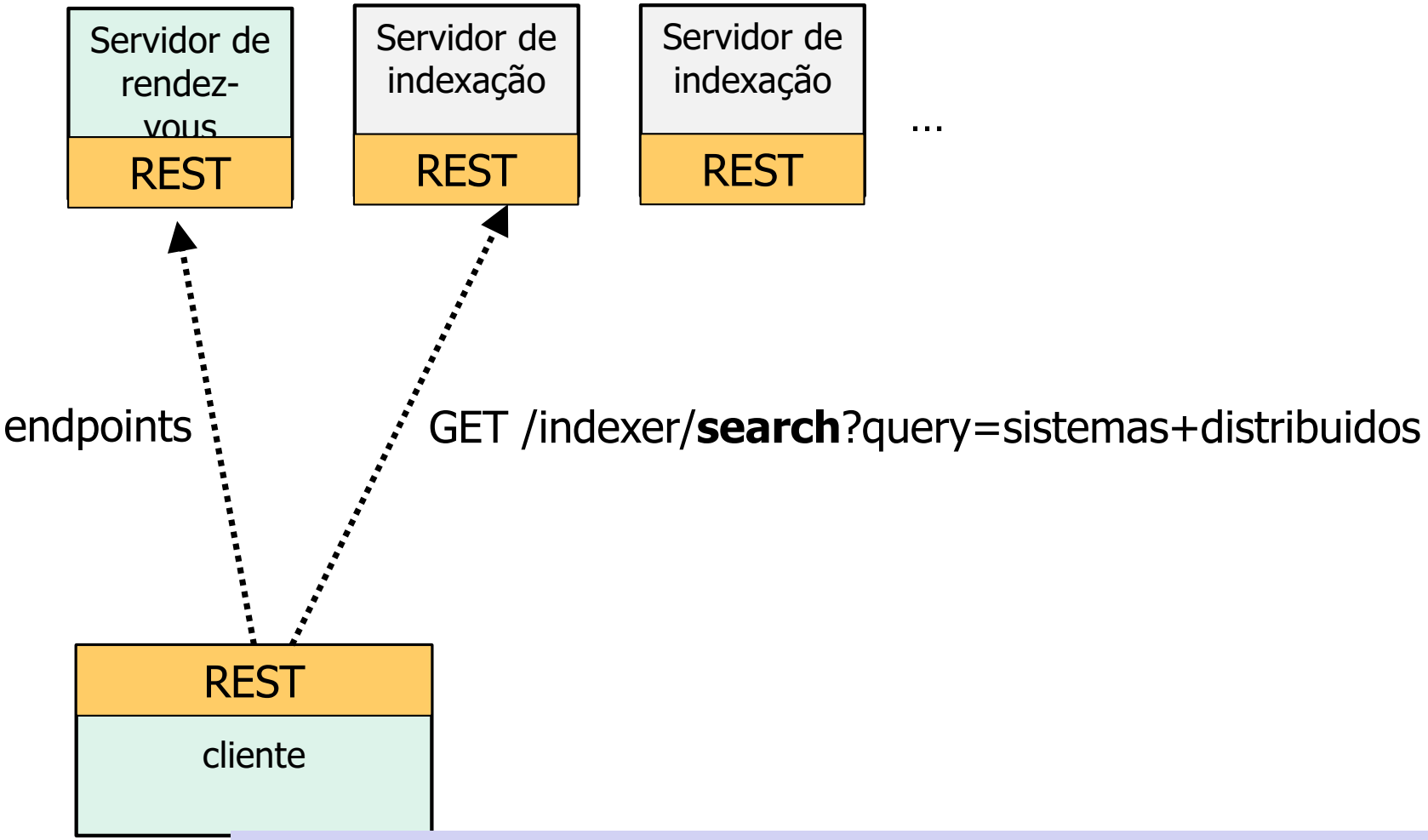

```
@POST
@Path("/{id}")
@Consumes(MediaType.APPLICATION_JSON)
void add(@PathParam("id") String id, Document doc);
```

Trabalho prático – Pesquisa




```
@DELETE
@Path("/{id}")
void remove(@PathParam("id") String id);
```

Trabalho prático – Pesquisa




```
@GET
@Path("/search")
@Produces(MediaType.APPLICATION_JSON)
List<String> search(@QueryParam("query") String keywords);
```

BIBLIOTECA PARA INDEXAR INFORMAÇÃO LOCALMENTE

```
public interface Storage {

    // Retrieves the list of urls of the documents that
    // have all the keywords being searched
    List<Document> search( List<String> keywords );

    //Adds the given document to the stored index
    boolean store( String id, Document doc );

    //Removes the given document from the index.
    boolean remove( String id );
}
```

BIBLIOTECA PARA INDEXAR INFORMAÇÃO LOCALMENTE

```
//criar storage
LocalVolatileStorage store = new LocalVolatileStorage();

// adicionar documento
String id = ...
Document doc = ...
boolean ok = store.store(id, doc);

// remover documento
String id = ...
boolean ok = store.remove(id);
```

BIBLIOTECA PARA INDEXAR INFORMAÇÃO LOCALMENTE

```
// pesquisar documentos
List<String> keywords = ...
List<Document> docs = store.search(keywords);
```

AGENDA

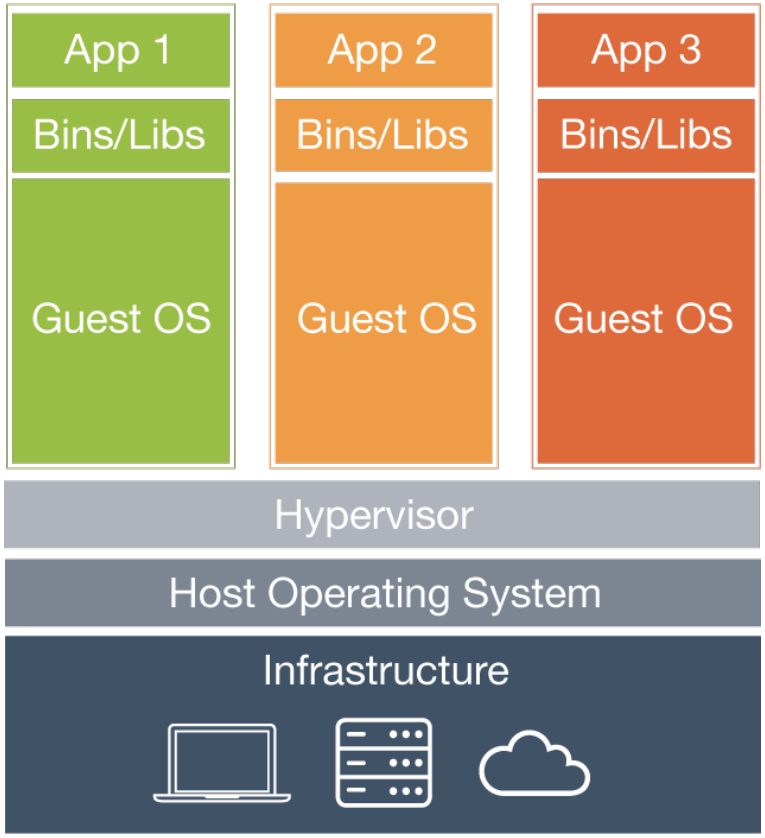
Apresentação do trabalho

Deployment usando Docker

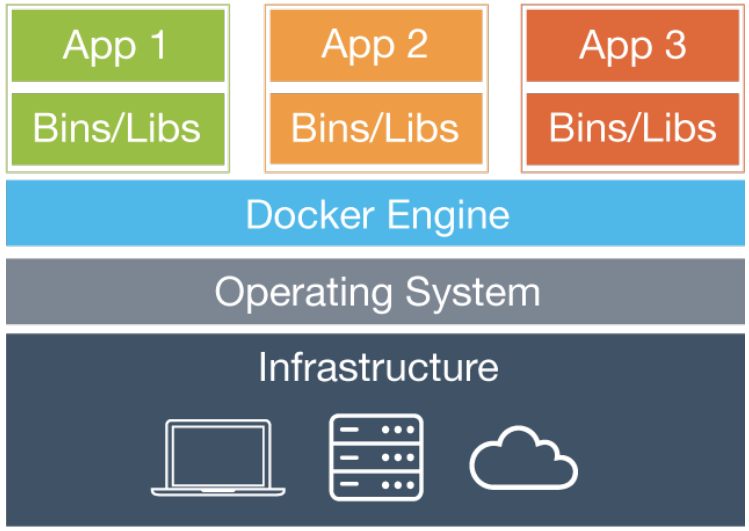
CONTAINER

Objetivo: permitir executar aplicações, incluindo todas as suas dependências

Máquina Virtual



Container



Images from docker.com

CRIAR IMAGEM DOCKER DO TRABALHO: PLUGINS

docker-maven-plugin

Permite criar e instalar imagem docker usando o maven

https://github.com/spotify/docker-maven-plugin

maven-dependency-plugin

Permite copiar todas as dependências para uma diretoria (que depois é usada pelo plugin do docker)

CRIAR IMAGEM DOCKER DO TRABALHO

Usar "pom.xml" que está na página da aula 2

mvn package

Cria imagem e instala no docker local. Nome da imagem: sd-work-username

Para alterar nome da imagem:

```
<plugin>
    <groupId>com.spotify</groupId>
    <artifactId>docker-maven-plugin</artifactId>
    …
    <configuration>
      <imageName>sd-work-${user.name}</imageName>
    …
    </configuration>
 …
</plugin>
```

EXECUTAR PROGRAMA NO CONTAINER

```
docker run sd-work-xpto java -cp /home/sd/* rest.server.RendezVousServer
```

imagem: sd-work-xpto

jars na diretoria: /home/sd/*

programa Java: rest.server.RendezVousServer

```
10-22-106-43:SD2017-P1-REST nmp$ docker run sd-work-nmp java -cp /home/sd/* rest.server.RendezVousServer
REST RendezVous Server ready @ http://0.0.0.0:8080/ : local IP = 172.17.0.2
```

MAIS INFORMAÇÃO

Como saber IP do container?

Solução simples: imprimir o IP quando se inicia o servidor

```
System.err.println("REST RendezVous Server ready @ " + baseUri + " :
local IP = " + InetAddress.getLocalHost().getHostAddress());
```

MAIS INFORMAÇÃO

Como aceder ao container a partir da máquina local?

Mapeando portos usando a opção –p porto_host:porto_container

```
docker run -p 8080:8080 sd-work-xpto java -cp /home/sd/*
rest.server.RendezVousServer
```