dgroove
Tecnologia emozionale.

ICT Training Center

Il tuo partner per la Formazione e la Trasformazione digitale della tua azienda

Note

Spring AI

Generative Artificial Intelligence con Java

Simone Scannapieco

Corso avanzato per Venis S.p.A, Venezia, Italia

Novembre 2025

Note

Docker Model Runner

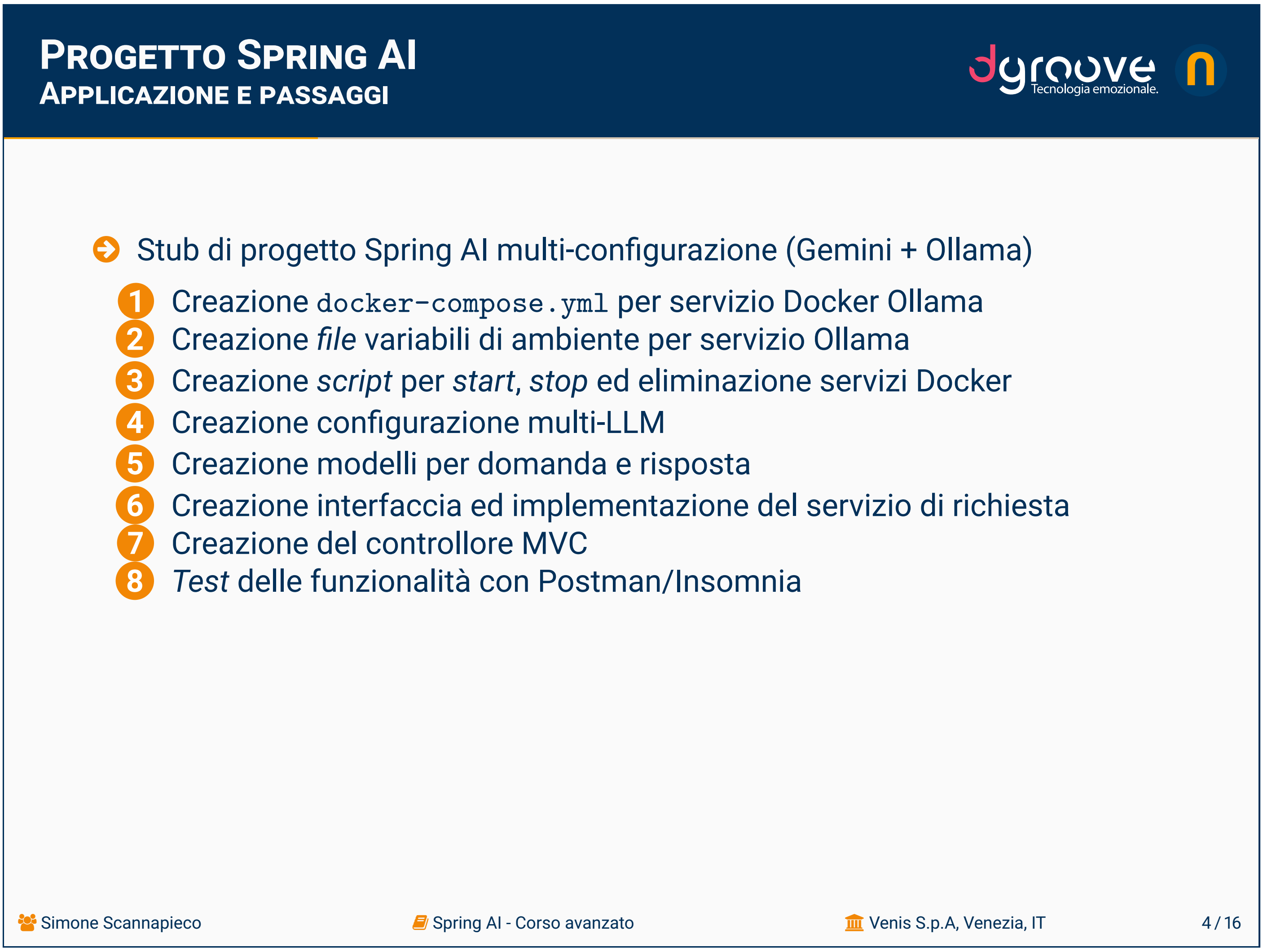
Note

Docker Model Runner
Descrizione

- Risposta di Docker ad Ollama
  - LLM in Docker *container* locali
  - Modelli AI generici dockerizzabili (WIP)
  - Docker mette a disposizione una serie di modelli *open source* scaricabili tramite Engine o Desktop

- `https://www.docker.com/blog/run-llms-locally/`
- `https://www.docker.com/blog/introducing-docker-model-runner/`

Note



Progetto Spring AI
Applicazione e passaggi

- Stub di progetto Spring AI multi-configurazione (Gemini + Ollama)
  1. Creazione `docker-compose.yml` per servizio Docker Ollama
  2. Creazione *file* variabili di ambiente per servizio Ollama
  3. Creazione *script* per *start*, *stop* ed eliminazione servizi Docker
  4. Creazione configurazione multi-LLM
  5. Creazione modelli per domanda e risposta
  6. Creazione interfaccia ed implementazione del servizio di richiesta
  7. Creazione del controllore MVC
  8. *Test* delle funzionalità con Postman/Insomnia

Note

Progetto Spring AI

Servizio Docker Ollama - I

File `docker-compose.yml`

```yaml
services:
  spring-ai-llm-gpu:
    image: ollama/ollama:${OLLAMA_VERSION:-latest}
    hostname: spring-ai-llm
    container_name: spring_ai_llm
    environment:
      OLLAMA_HOST: "${OLLAMA_HOST:-0.0.0.0}:${OLLAMA_PORT-11434}"
      OLLAMA_DEBUG: ${OLLAMA_DEBUG:-false}
      OLLAMA_FLASH_ATTENTION: ${OLLAMA_FLASH_ATTENTION:-false}
      OLLAMA_KEEP_ALIVE: ${OLLAMA_KEEP_ALIVE:-"5m"}
      OLLAMA_MAX_LOADED_MODELS: ${OLLAMA_MAX_LOADED_MODELS:-1}
      OLLAMA_NUM_PARALLEL: ${OLLAMA_NUM_PARALLEL:-1}
    expose:
      - ${OLLAMA_PORT:-11434}
    deploy:
      resources:
        reservations:
          devices:
            - driver: nvidia
              count: all
              capabilities: [gpu]
    volumes:
      - spring_ai_llm:/root/.ollama
    restart: unless-stopped
    profiles: [llm-gpu]

...
```

Note

Progetto Spring AI
Servizio Docker Ollama - II

File `docker-compose.yml`

```
...

  spring-ai-llm-cpu:
    image: ollama/ollama:${OLLAMA_VERSION:-latest}
    hostname: spring-ai-llm
    container_name: spring_ai_llm
    environment:
      OLLAMA_HOST: "${OLLAMA_HOST:-0.0.0.0}:${OLLAMA_PORT-11434}"
      OLLAMA_DEBUG: ${OLLAMA_DEBUG:-false}
      OLLAMA_FLASH_ATTENTION: ${OLLAMA_FLASH_ATTENTION:-false}
      OLLAMA_KEEP_ALIVE: ${OLLAMA_KEEP_ALIVE:-"5m"}
      OLLAMA_MAX_LOADED_MODELS: ${OLLAMA_MAX_LOADED_MODELS:-1}
      OLLAMA_NUM_PARALLEL: ${OLLAMA_NUM_PARALLEL:-1}
    expose:
      - ${OLLAMA_PORT:-11434}
    volumes:
      - spring_ai_llm:/root/.ollama
    restart: unless-stopped
    profiles: [llm-cpu]

volumes:
  spring_ai_llm:
    name: spring_ai_llm
```

Note

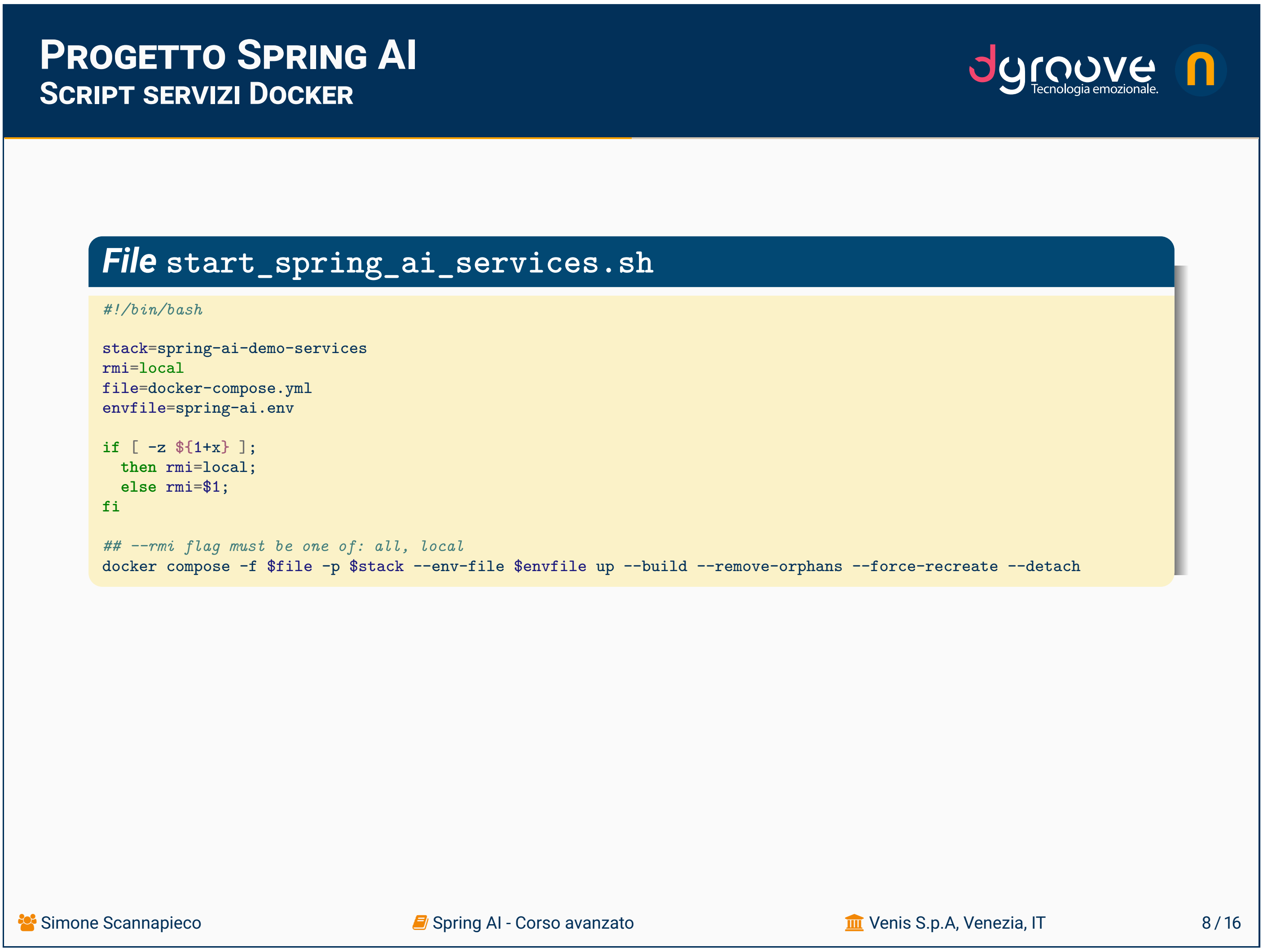
Progetto Spring AI

Variabili di ambiente servizio Docker Ollama

File spring-ai.env

```
# MODE:
# "llm-cpu" --> Large Language Model in cpu mode
# "llm-gpu" --> Large Language Model in gpu mode
MODE=llm-cpu
COMPOSE_PROFILES=${MODE}
LOG_LEVEL=WARNING                                                                    # default: WARNING

# Ollama configuration
#OLLAMA_VERSION=0.1.39                                                                # default: latest
OLLAMA_HOST=spring-ai-llm                                                            # default: 0.0.0.0
OLLAMA_PORT=11434                                                                      # default: 11434
OLLAMA_DEBUG=false                                                                     # default: false
OLLAMA_FLASH_ATTENTION=false                                                           # default: false
OLLAMA_KEEP_ALIVE="5m"                                                                  # default: "5m"
OLLAMA_MAX_LOADED_MODELS=2                                                                 # default: 1
OLLAMA_NUM_PARALLEL=1                                                                      # default: 1
```

Note



Progetto Spring AI
Script servizi Docker

File `start_spring_ai_services.sh`

```bash
#!/bin/bash

stack=spring-ai-demo-services
rmi=local
file=docker-compose.yml
envfile=spring-ai.env

if [ -z ${1+x} ];
  then rmi=local;
  else rmi=$1;
fi

## --rmi flag must be one of: all, local
docker compose -f $file -p $stack --env-file $envfile up --build --remove-orphans --force-recreate --detach
```

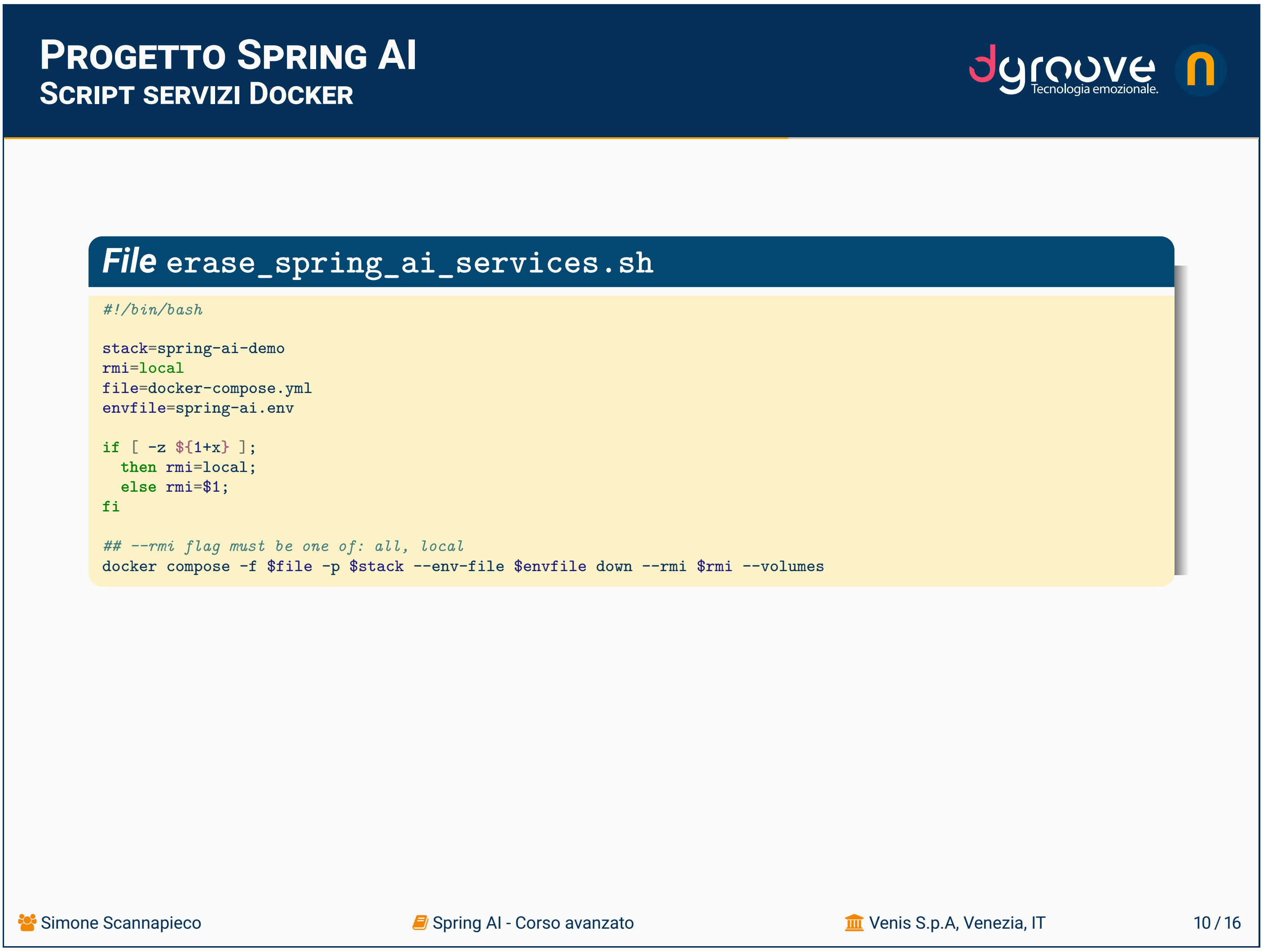
Note

Progetto Spring AI
Script servizi Docker

File `stop_spring_ai_services.sh`

```bash
#!/bin/bash

stack=spring-ai-demo-services
rmi=local
file=docker-compose.yml
envfile=spring-ai.env

if [ -z ${1+x} ];
  then rmi=local;
  else rmi=$1;
fi

## --rmi flag must be one of: all, local
docker compose -f $file -p $stack --env-file $envfile down --rmi $rmi
```

Note



Progetto Spring AI
Script servizi Docker

File `erase_spring_ai_services.sh`

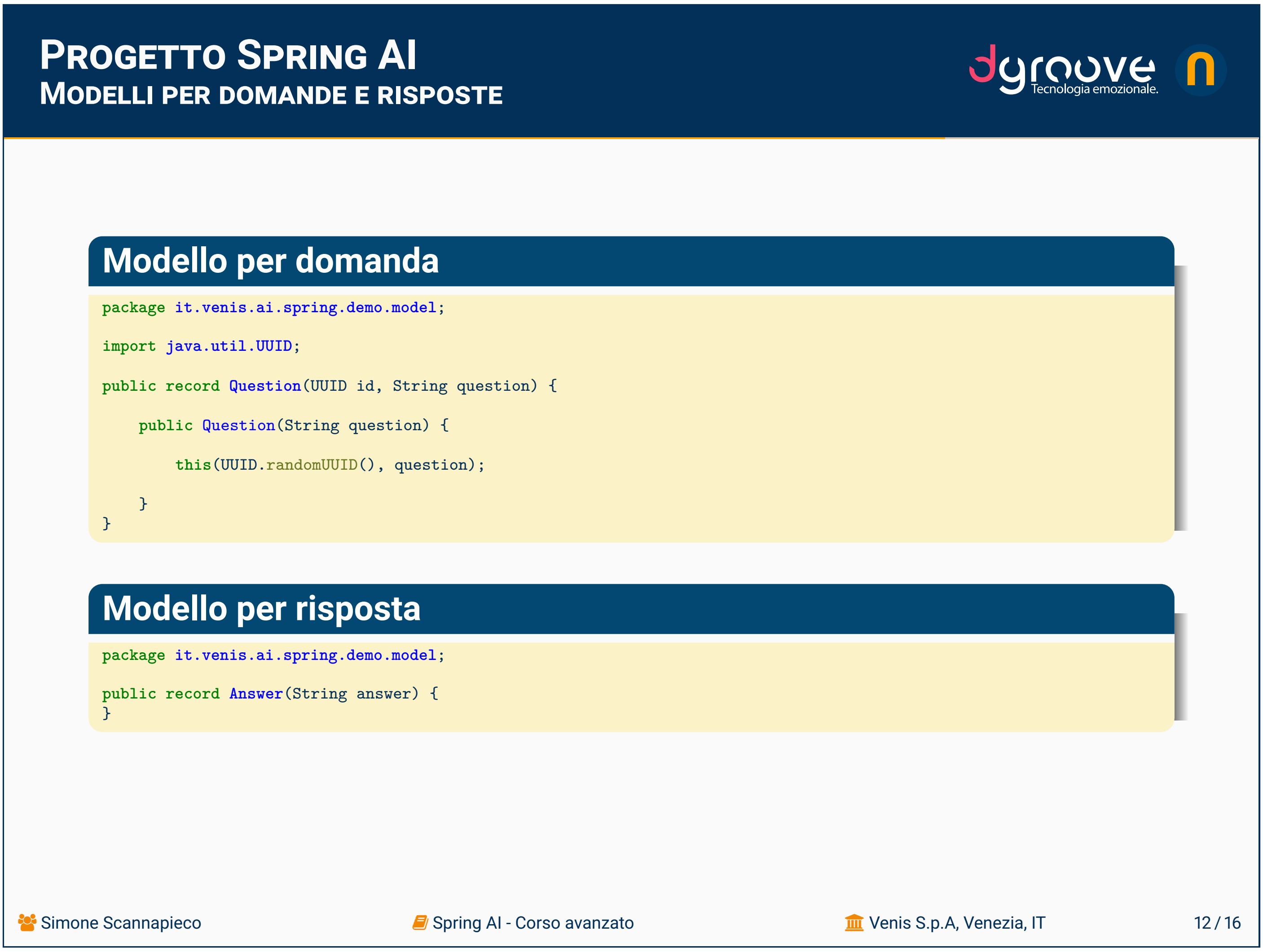
```bash
#!/bin/bash

stack=spring-ai-demo
rmi=local
file=docker-compose.yml
envfile=spring-ai.env

if [ -z ${1+x} ];
  then rmi=local;
  else rmi=$1;
fi

## --rmi flag must be one of: all, local
docker compose -f $file -p $stack --env-file $envfile down --rmi $rmi --volumes
```

Note

Configurazione Gemini + Ollama

```java
package it.venis.ai.spring.demo.config;

import org.springframework.ai.chat.client.ChatClient;
import org.springframework.ai.ollama.OllamaChatModel;
import org.springframework.ai.openai.OpenAiChatModel;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Primary;

@Configuration
public class ChatClientConfig {

    @Bean
    @Primary
    public ChatClient openAiChatClient(OpenAiChatModel openaiChatModel) {
        return ChatClient.create(openaiChatModel);
        /*
         * or:
         * ChatClient.Builder chatClientBulder = ChatClient.builder(openaiChatModel);
         * return chatClientBulder.build();
         */
    }

    @Bean
    public ChatClient ollamaChatClient(OllamaChatModel ollamaChatModel) {
        ChatClient.Builder chatClientBulder = ChatClient.builder(ollamaChatModel);
        return chatClientBulder.build();
        /*
         * or:
         * return ChatClient.create(ollamaChatModel);
         */
    }

}
```

Note



Progetto Spring AI
Modelli per domande e risposte

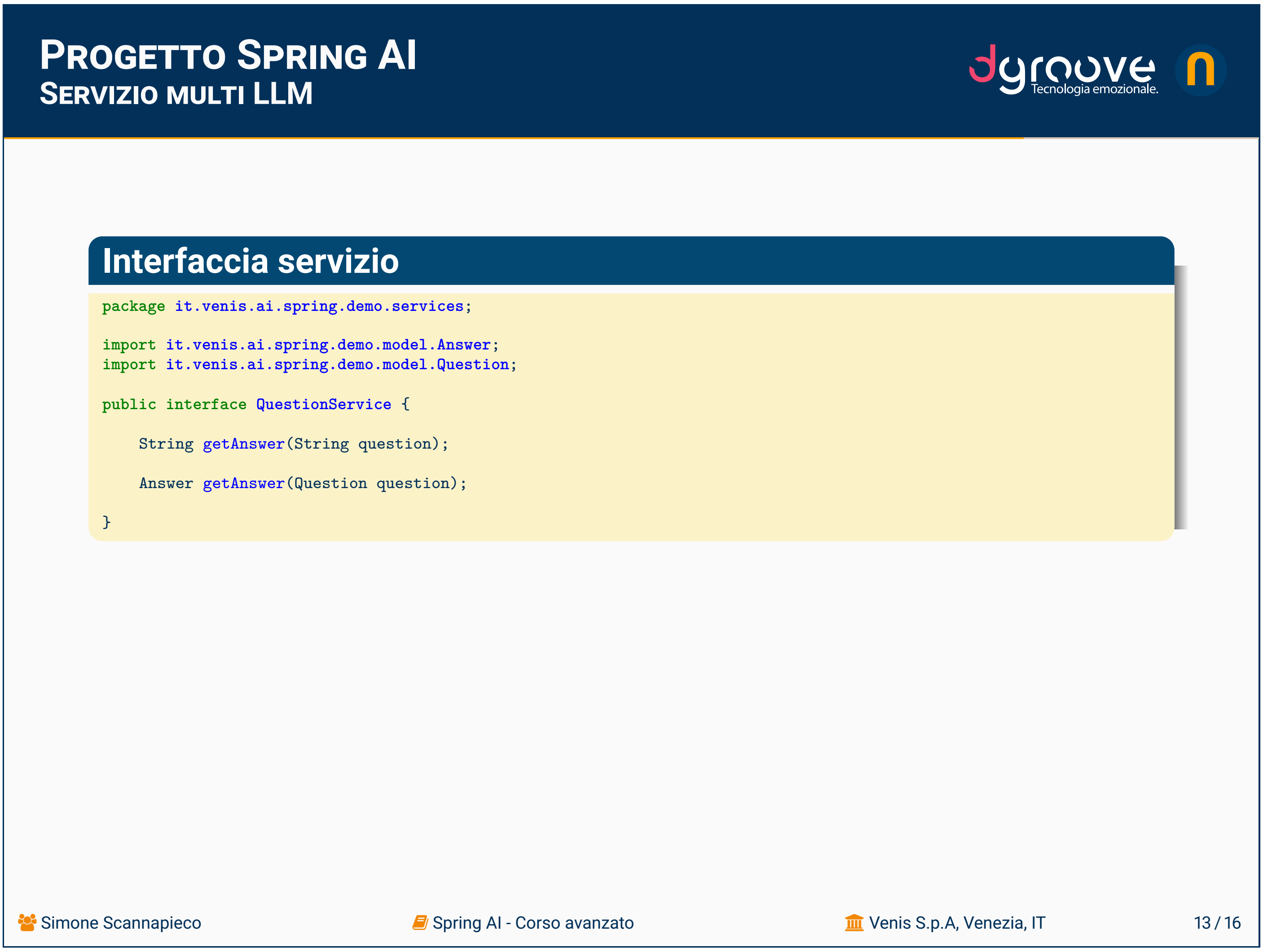
Modello per domanda

```
package it.venis.ai.spring.demo.model;

import java.util.UUID;

public record Question(UUID id, String question) {

    public Question(String question) {

        this(UUID.randomUUID(), question);

    }
}
```

Modello per risposta

```
package it.venis.ai.spring.demo.model;

public record Answer(String answer) {
}
```

Note



Progetto Spring AI
Servizio multi LLM

Interfaccia servizio

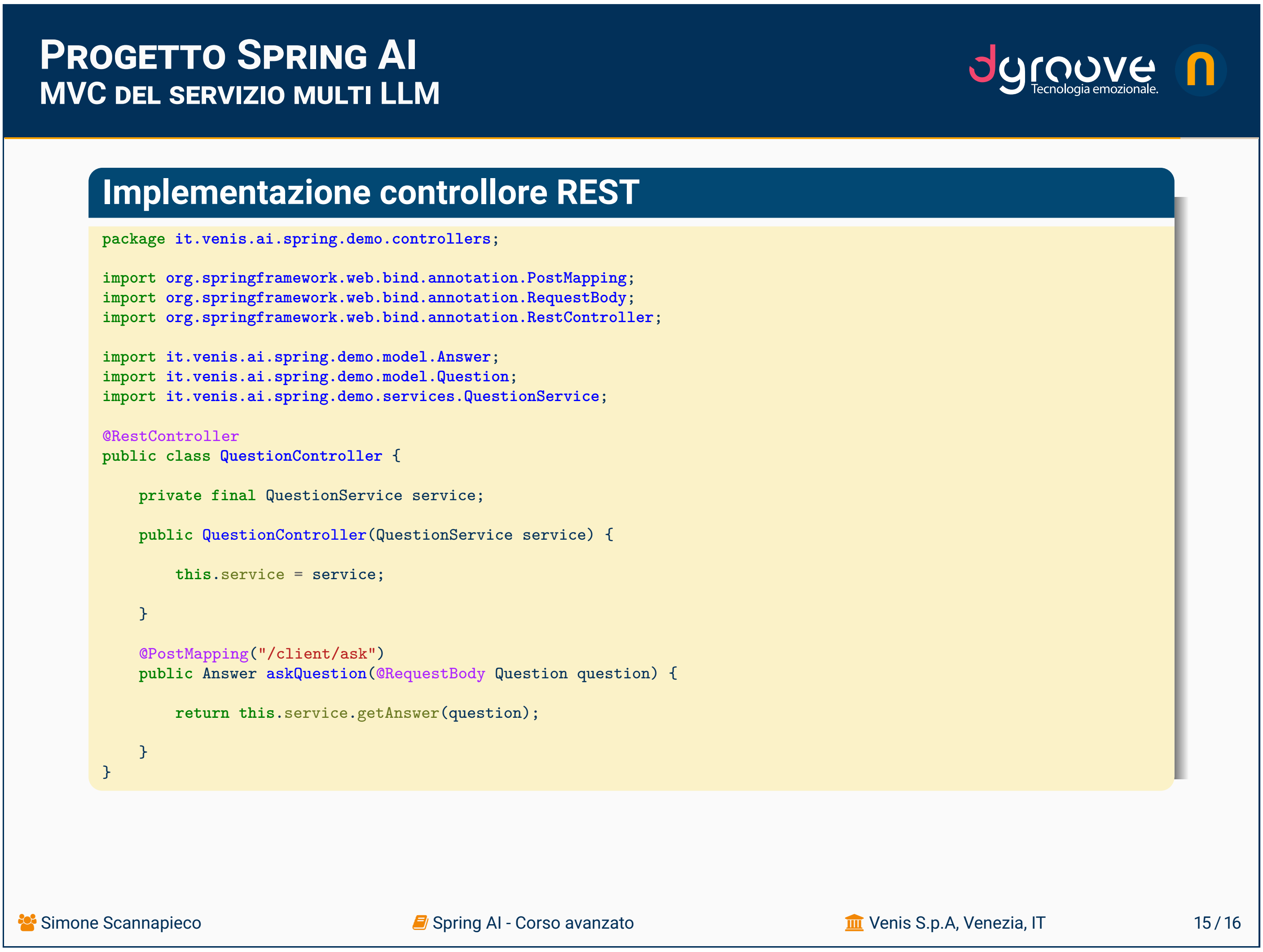
```java
package it.venis.ai.spring.demo.services;

import it.venis.ai.spring.demo.model.Answer;
import it.venis.ai.spring.demo.model.Question;

public interface QuestionService {

    String getAnswer(String question);

    Answer getAnswer(Question question);

}
```

Note

Implementazione servizio

```java
package it.venis.ai.spring.demo.services;

import org.springframework.ai.chat.client.ChatClient;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.context.annotation.Configuration;
import org.springframework.stereotype.Service;

import it.venis.ai.spring.demo.model.Answer;
import it.venis.ai.spring.demo.model.Question;

@Service
@Configuration
public class QuestionServiceImpl implements QuestionService {

    private final ChatClient chatClient;

    public QuestionServiceImpl(@Qualifier("ollamaChatClient") ChatClient chatClient) {
        this.chatClient = chatClient;
    }

    @Override
    public String getAnswer(String question) {
        return this.chatClient.prompt()
                .user(question)
                .call()
                .content();
    }

    @Override
    public Answer getAnswer(Question question) {
        return new Answer(getAnswer(question.question()));
    }

}
```

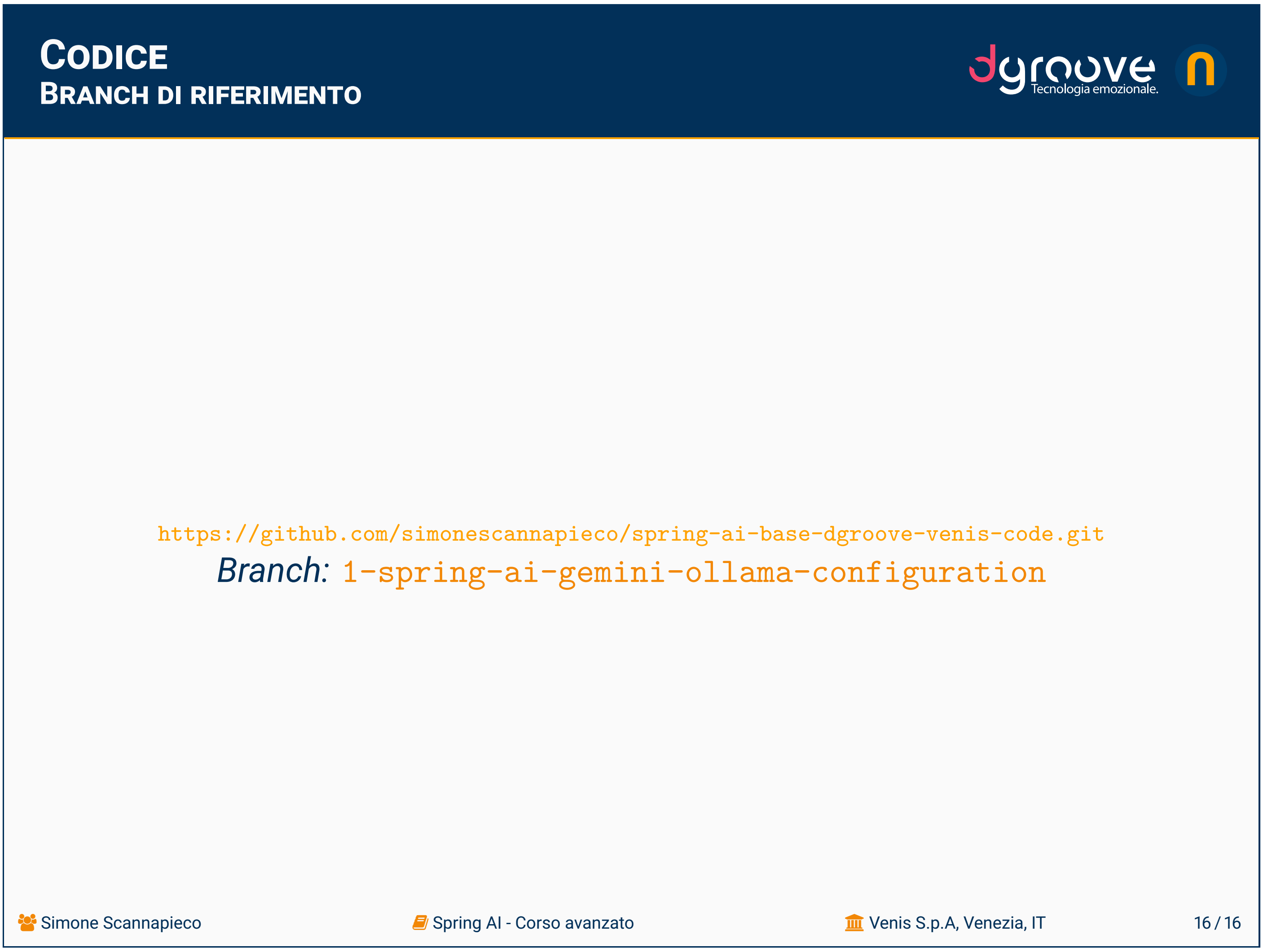
Note



Implementazione controllore REST

```java
package it.venis.ai.spring.demo.controllers;

import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;

import it.venis.ai.spring.demo.model.Answer;
import it.venis.ai.spring.demo.model.Question;
import it.venis.ai.spring.demo.services.QuestionService;

@RestController
public class QuestionController {

    private final QuestionService service;

    public QuestionController(QuestionService service) {

        this.service = service;

    }

    @PostMapping("/client/ask")
    public Answer askQuestion(@RequestBody Question question) {

        return this.service.getAnswer(question);

    }
}
```

Note



Codice
Branch di riferimento

`https://github.com/simonescannapieco/spring-ai-base-dgroove-venis-code.git`

Branch: `1-spring-ai-gemini-ollama-configuration`

Note