SPRING INITIALIZER CHEAT SHEET

CORE

AOP: AOP that stands for Aspect-Oriented Programming is a programming paradigm to make cross-cutting concerns of an application, such as transaction management, logging, security, and data transfer modular. Cross-cutting concerns are separated from the business logic as aspects. An aspect performs action (advice) on a point, also known as join point during the execution of a program. Spring AOP is a framework that brings in AOP style of programming to applications using either a schema-based approach or the @AspectJ annotation style.

https://docs.spring.io/spring/docs/current/spring-framework-reference/html/aop.html

Atomikos (JPA): A popular open source transaction manager. Using this dependency, you can embed Atomikos into your Spring Boot application. When you add the Atomikos dependency in your application, Spring Boot auto-configures Atomikos and applies

appropriate settings to your Spring beans to perform startup and shutdown of the transaction manager in the correct ordering.

https://www.atomikos.com/Documentation/SpringIntegration

Bitronix (JTA): A popular open source transaction manager that Spring Boot supports for distributed JTA transactions that may span multiple transactional resources. In order to communicated with multiple transactional resources, the embedded Bitronix transaction manager (BTM) is built on the X/Open XA specification and fully compliant with the JTA 1.1 API.

https://github.com/bitronix/btm/wiki/Overview

Cache: As apparent from the name, this starter allows you to quickly add caching to your application. Cache provides an abstraction that allows consistent use of various caching solutions, such as JCache, EhCache 2.x, Gemfire cache, Caffeine, and Guava with minimal impact on the code out of the box.

https://docs.spring.io/spring/docs/current/spring-framework-reference/html/cache.html

Configuration Processor: Generate metadata for your custom configuration properties. Metadata provide details of all supported configuration properties. Using the metadata, IDE editors offer contextual help and “code completion” as you are working with configuration files, such as application.properties or application.yml files.

http://docs.spring.io/spring-boot/docs/current/reference/html/configuration-metadata.html

DevTools: A module introduced in Spring Boot 1.3 to improve the development-time when working on Spring Boot applications. One key feature that DevTools brings in is to set property defaults for development. For example, caching, which is performance intensive is disabled by default during development and h2 console is enabled by default for a quick peek to h2-stored data. DevTools also automatically restarts your application when any classpath file changes and can be used to trigger a browser refresh when a resource is changed. Additionally, DevTools supports remote application updates and restarts.

http://docs.spring.io/spring-boot/docs/current/reference/html/using-boot-devtools.html

Lombok: A Java annotation library which helps to code faster by generating boilerplate code. You can add Lombok annotations in your source file, and Lombok will inject code based on the annotation, which will be immediately available to you. Simple examples are @Getter and @Setter annotations that generates a getter and setter for a field, respectively. The most useful and frequently used Lombok annotation is @Data. This annotation generates overridden code for toString(), hashCode(), equals(), and also getter and setter methods for the fields.

http://jnb.ociweb.com/jnb/jnbJan2010.html

Narayana (JTA): A popular open source JTA transaction manager

implementation supported by JBoss. Narayana was introduced as a starter dependency in Spring Boot 1.4.0.M2. With this dependency in your project, Spring Boot will automatically configure Narayana and post-process your beans to ensure that startup and shutdown ordering is correct. With Narayana as the transaction manager in your Spring Boot application, you can simply use JTA or Spring annotations to manage the transactions of your application.

http://jbossts.blogspot.in/2016/06/narayana-in-spring-boot.html

Retry: Provides declarative retry support for Spring applications in Spring Batch, Spring Integration, Spring for Apache Hadoop, amongst others. With this dependency, you can have methods annotated with @Retryable. Such retryable methods are called like any other methods, but, whenever the method fails with an exception, Spring will automatically retry to call the method up to the default three times. This dependency is must-have in your application if you have to interact with external resources. Typical use cases include remote call to a web service or RMI service that fails because of a network glitch. Also, a database update 1DeadLockLoserException that may get resolved after a short wait.

http://www.mscharhag.com/spring/spring-retry

Security: A powerful and highly customizable authentication and access-control framework. With Spring Security, you can implement authentication with different providers, such as DAO, LDAP, Single Sign-On (SSO), OpenID, and OAuth 2.0 and also fine-grained role-based authorization. Spring Security also protects your application against

attacks like session fixation, clickjacking, cross site request forgery, and several other application security threats.

https://projects.spring.io/spring-security/

Session: API and implementations for managing a user's session information. Spring Session makes it easy to write horizontally scalable cloud native applications. You can have specialized external session stores, such as Redis or Apache Geode Storage to store the session state. Spring Session also allows keeping the HttpSession alive when users are making requests over Web Socket and allows multiple sessions per browser. Spring Session also allows access to session data from non-web request processing code, such as JMS message processing code. In addition, with Spring Session you can write Restful API's that can extract the session id from an HTTP header rather than relying on cookies.

http://projects.spring.io/spring-session/

Validation: Java Bean Validation (JSR-303) infrastructure that allows validation rules to be specified directly in the code they are intended to validate, instead of creating validation rules in separate classes. Spring validation is also well integrated in Spring MVC and therefore can be effortlessly used to address the validation requirements of your Web application.

http://docs.spring.io/spring/docs/current/spring-framework-reference/html/validation.html

Web

Web: Enables full-stack web development with Spring Web and Spring MVC. This Web starter brings in several dependencies for Web development, such as Tomcat, Hibernate validator, and Jackson data bind, which in turn brings in additional dependencies, such as Spring AOP, Context, Expression, and JBoss logging.

https://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/#boot-features-developing-web-applications

Websocket: Adds capability of Web socket for two-way communication between client and server using the SockJS and STOMP protocols. The Websocket module is compatible with the Java WebSocket API standard (JSR-356) and also provides additional value-add. Some uses cases for WebSocket are web applications in finance, games, collaboration, and others where the client and server need to exchange events at high frequency and with low latency.

http://docs.spring.io/spring/docs/current/spring-framework-reference/html/websocket.html

Web Services: Also, known as Spring-WS allows creation of document-driven contract-first SOAP service. Spring-WS uses one of the many JAXP APIs such as DOM, SAX, StAX, JDOM, dom4j, XOM, and even marshalling technologies, such as JAXB, Castor, XMLBeans, JiBX, and XStream. to manipulate XML payloads. Spring-WS is based on Spring itself, and therefore Spring concepts such as dependency injection are available as an integral part of your Web service. Spring-WS enforces

best practices such as the WS-I basic profile, Contract-First development, and having a loose coupling between contract and implementation.

http://projects.spring.io/spring-ws/

Jersey (JAX-RS): A RESTful Web Services framework to build RESTful APIs using JAX-RS specification and easily deploy it to Tomcat or any other Spring's Boot supported container. Using Jersey, you write JAX-RS endpoint classes as Spring beans in order to be able to use Spring DI inside the JAX-RS endpoints.

https://dzone.com/articles/using-jax-rs-with-spring-boot-instead-of-mvc

Ratpack: Ratpack is a framework built on the highly performant and efficient Netty event-driven networking engine. The Ratpack framework facilitate fast, efficient, evolvable and well tested HTTP applications. The Ratpack Spring Boot dependency provides Spring Boot integration for the Ratpack framework. It allows the Ratpack server registry to be created from a Spring ApplicationContext, and Ratpack itself to be embedded in a Spring Boot application.

https://ratpack.io/manual/current/spring.html

Vaadin: Vaadin is an open-source Java web UI framework for rich Internet applications. The Spring Boot Vaadin dependency allows managing rich Vaadin UI with the power of Spring. With Vaadin integrated with Spring Boot, you can quickly get up and running a fully

configured environment, including a standalone server. You can also start adding your content to the Vaadin UI class right away.

https://vaadin.com/spring

Rest Repositories: Allows Spring Data repositories to be exposed as REST services. With Rest Repositories, you can export JPA entities as REST endpoints. Rest Repositories analyzes your the domain model of your application and exposes hypermedia-driven HTTP resources for aggregates contained in the model.

http://docs.spring.io/spring-data/rest/docs/current/reference/html/

HATEOAS: HATEOAS that stands for Hypermedia as the Engine of Application State, is a constraint of the REST application architecture that decouples client and server in a way that the server can evolve independently. As per HATEOS, a REST client needs no prior knowledge about how to interact with any particular application or server beyond a generic understanding of hypermedia. Spring HATEOS enables creating REST representations that follow the HATEOAS principle when working with Spring and especially Spring MVC.

https://spring.io/guides/gs/rest-hateoas/

Rest Repositories HAL Browser: Allows browsing Spring Data REST repositories in your browser. The Rest Repositories HAL Browser is a web app that starts a HAL-powered JavaScript. You can point it at any Spring Data REST API and use it to navigate the app and create new resources.

http://api.opensupporter.org/hb2/browser.html#/api/v1

Mobile: A framework that simplifies the development of mobile web applications. Spring Mobile is an extension to Spring MVC that that provides capabilities to detect the type of device making a request to your Spring web site and serve alternative views based on that device.

http://projects.spring.io/spring-mobile/

REST Docs: Module to document RESTful services by combining hand-written and auto-generated snippets produced with Spring MVC Test or REST Assured. Spring REST Docs allows you to work with details of the HTTP requests that it consumes and the HTTP responses that it produces shielding your documentation from the inner-details of the service's implementation. By default, Spring REST Docs uses Asciidoctor, but can also be configured to use Markdown.

http://docs.spring.io/spring-restdocs/docs/current/reference/html5/

TEMPLATE ENGINES

Freemarker: FreeMarker is a Java-based template engine for both standalone and Web-based applications. Freemarker is commonly used to generate text output for HTML web pages, e-mails, configuration files, and source code. Templates are written in text files using the FreeMarker Template Language (FTL). Spring Boot Freemarker integrates the FreeMarker templating engine with Spring Boot applications.

https://hellokoding.com/spring-boot-hello-world-example-with-freemarker/

Velocity: Velocity is a free open-source templating engine. Its clean separation of template and Java code makes it suitable for Web development using the Model-View-Controller (MVC) pattern. The Velocity Template Language (VTL) is the templating language that provides the easiest and cleanest way to incorporate dynamic content in a web page. Spring Boot Velocity integrates the Velocity templating engine with Spring Boot applications.

With the deprecation of Velocity support in Spring Framework 4.3, Velocity in Spring Boot is supported between Spring Boot 1.1.6.RELEASE and 1.4.0.M2

http://velocity.apache.org/

Groovy Templates: Introduced in Spring Boot 1.1.0.M2, Spring Boot Groovy Templates supports the new template engine that Groovy 2.3 provides. The Groovy template engine brings in an innovative templating system based on the builder syntax with hierarchical (builder) syntax to generate XML-like contents, particularly targeting HTML5. Spring Boot Groovy Templates integrates the Groovy templating engine with Spring Boot applications.

http://docs.groovy-lang.org/latest/html/documentation/template-engines.html

Thymeleaf: Thymeleaf is a server-side Java template engine for both web and standalone environments. Thymeleaf brings in elegant natural

templates with which HTML can be correctly displayed in browsers and also work as static prototypes. Spring Boot Thymeleaf integrates the Thymeleaf templating engine with Spring Boot applications.

https://springframework.guru/spring-boot-web-application-part-2-using-thymeleaf/

Mustache: Mustache is a web template system with implementations available for a wide set of programming languages, including Java with JMustache. Mustache lacks any explicit control flow statements, like if and else conditionals or for loops, and therefore is described as a "logic-less" system. However, both looping and conditional evaluation can be achieved using section tags processing lists and lambdas. Spring Boot 1.2.2 introduced this Mustache module to provide support for JMustache in Spring Boot application.

https://spring.io/blog/2016/11/21/the-joy-of-mustache-server-side-templates-for-the-jvm

SQL

JPA: Java Persistence API (JPA) provides Java developers with an object/relational mapping facility for managing relational data in Java applications. JPA consists of four areas: The persistence API, query language, Java Persistence Criteria API, and Object/relational mapping metadata. Spring Boot JPA brings in Spring Data JPA, Spring ORM, and Hibernate to work with relational data as entities.

https://spring.io/guides/gs/accessing-data-jpa/

JOOQ: JOOQ that stands for Java Object Oriented Querying is a code generation and SQL data access library for Java. JOOQ embeds SQL as an internal domain-specific language directly in Java, making it easy for developers to write and read code that almost feels like actual SQL. Spring Boot JOOQ provides support for using JOOQ in Spring Boot application.

http://docs.spring.io/spring-boot/docs/1.3.8.RELEASE/reference/html/boot-features-jooq.html

MyBatis: MyBatis is a Java persistence framework with support for custom SQL, stored procedures and advanced mappings. MyBatis automates the mapping between SQL databases and Java objects using an XML descriptor or annotations. Spring Boot MyBatis provides persistence support using MyBatis in Spring Boot applications.

http://www.mybatis.org/spring-boot-starter/mybatis-spring-boot-autoconfigure/

JDBC: Provides an abstraction on top of JDBC API using JdbcTemplate. Spring Boot JDBC also provides great transaction management capabilities using annotation based approach through PlatformTransactionManager (DataSourceTransactionManager).

https://spring.io/guides/gs/relational-data-access/

H2: H2 is a light, fast, and easy to use Java database that can be used both in embedded and server mode. H2 is often used to emulate production databases, such as Oracle, MySQL, and Postgres during development. H2 ships with a web based database console, which you

can use while your application is under development. It is a convenient way to view the database tables created by your application and run queries against them in memory database. Out of the box, the Spring Boot H2 module makes H2 Database very easy to use in . If the H2 database is found on your classpath, Spring Boot will automatically set up an in memory H2 database for your use.

https://springframework.guru/using-the-h2-database-console-in-spring-boot-with-spring-security/

HSQLDB: Hyper SQL Database (HSQLDB) is a relational database management system (RDBMS) written in Java. To provide a fast, multithreaded, and transactional database engine with in-memory and disk-based tables, HSQLDB supports both embedded and server modes. Spring Boot auto configures an embedded HSQLDB instance. You don't even need to provide any connection URLs. Simply include this dependency in your application.

https://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-sql.html

Apache Derby: A relational database management system (RDBMS) developed by the Apache Software Foundation. With a small footprint of about 2.6 megabytes for the base engine and embedded JDBC driver, Derby makes a popular choice as an embedded database engine during development.

https://db.apache.org/derby/

MySQL: Provides MySQL Connector/J, the official JDBC driver for MySQL

relational database management system (RDBMS). This driver is a Type 4 JDBC driver, which means that the driver is a pure Java implementation of the MySQL protocol and does not rely on the MySQL client libraries.

https://springframework.guru/configuring-spring-boot-for-mysql/

PostgreSQL: Provides PostgreSQL JDBC driver to support accessing a PostgreSQL database from a Spring Boot application. The driver is an open source Type 4 JDBC driver for PostgreSQL.

https://springframework.guru/configuring-spring-boot-for-postgresql/

SQL Server: Brings in Microsoft JDBC driver for SQL Server. This driver is a Type 4 JDBC driver that provides database connectivity to SQL Server through the standard JDBC application program interfaces (APIs). The Spring Boot SQL Server starter dependency has been made available from Spring Boot 1.5.0.RC1.

https://msdn.microsoft.com/en-us/library/mt484311(v=sql.110).aspx

Flyway: An open source database migration library that favors simplicity and convention over configuration. Spring Boot Flyway when included in your application will automatically autowire Flyway with its DataSource and invoke it on startup. You can then configure a good number of Flyway properties directly from your application.properties or application.yml file.

https://docs.spring.io/spring-boot/docs/current/reference/html/howto-database-initialization.html#howto-execute-flyway-database-migrations-on-startup

Liquibase: Liquibase is an open source database-independent library for tracking, managing, and applying database schema changes. With Spring Boot Liquibase, you can automatically run Liquibase database migrations on startup. You don't even need to define a bean for Liquibase. You only need to put your change log in db/changelog /db.changelog-master.yaml and Liquibase migrations will run automatically on startup.

http://docs.spring.io/spring-boot/docs/current/reference/html/howto-database-initialization.html

NOSQL

MongoDB: A document-based, schemaless NoSQL database designed for Agile development. Spring Data MongoDB provides integration with the MongoDB document database. Spring Data MongoDB provides a POJO centric model for interacting with a MongoDB collection. With Spring Data MongoDB, you can easily write a Repository style data access layer in your Spring Boot application.

http://projects.spring.io/spring-data-mongodb/

Cassandra: An open source distributed NoSQL database management system designed to handle large amounts of data. Spring Data for Cassandra provides a familiar and consistent Spring-based programming model for new datastores while retaining store-specific features and capabilities.

http://projects.spring.io/spring-data-cassandra/

Couchbase: A document database with a SQL-based query language. Couchbase provides developers to build applications easier and faster by leveraging the power of SQL with the flexibility of JSON. Spring Boot 1.4.0 MILESTONE 2 introduced a first-class integration of Couchbase into Spring Boot.

https://spring.io/blog/2016/04/14/couchbase-as-a-first-class-citizen-of-spring-boot-1-4

Neo4j: Neo4j is a NoSQL graph database ideal for creating graph database with the labeled property graph model. In this model, entities are represented as nodes. Each node can be tagged with labels and a node can have multiple properties. Nodes are associated with relationship. Spring Data Neo4J provides consistent APIs that do typical CRUD-like operations by executing queries. Typically, you have to annotate POJOs (Plain Old Java Objects) that map the domain objects to the underlying database.

https://spring.io/guides/gs/accessing-data-neo4j/

Redis: Redis is a cache, message broker, and key-value store. With Spring Data Redis, Spring Boot offers basic auto-configuration for the Jedis client library and abstractions on top of it. Using Spring Data Redis, you can easily configure and access Redis from Spring applications.

http://projects.spring.io/spring-data-redis/

Gemfire: GemFire is in-memory Data Grid powered by Apache Geode. Gemfire provides highly available distributed cache, elastic in-memory

computing, transaction processing, and event notification and processing. Spring Data GemFire simplifies building highly scalable Spring-powered applications using Pivotal GemFire as a distributed data management platform.

http://projects.spring.io/spring-data-gemfire/

Solr: Apache Solr is a standalone full-text search server. Internally, Solr uses the Lucene Java search library for full-text indexing and search. Solr also provides REST-like HTTP/XML and JSON APIs that make it usable from most popular programming languages. Spring Data for Apache Solr provides easy configuration and access to Apache Solr Search Server from Spring applications.

http://projects.spring.io/spring-data-solr/

Elasticsearch: Elasticsearch is a search and analytics engine. it provides a distributed, multitenant-capable full-text search engine with a RESTful web interface and schema-free JSON documents. Spring Data Elasticsearch contains a custom namespace allowing definition of repository beans as well as elements for instantiating a Elasticsearch server.

http://docs.spring.io/spring-data/elasticsearch/docs/current/reference/html/

CLOUD CORE

Cloud Connectors: Simplifies the process for Spring applications to

connect to services in cloud platforms such as Cloud Foundry and Heroku. Using Spring Cloud Connectors, applications running on cloud platforms can discover bound services and deployment information at runtime, and register discovered services as Spring beans.

http://cloud.spring.io/spring-cloud-connectors/

Cloud Bootstrap: Brings in a "bootstrap" context, which is a parent context for the main application. Out of the box, it is responsible for loading configuration properties from the external sources, and also decrypting properties in the local external configuration files.

http://projects.spring.io/spring-cloud/spring-cloud.html#_spring_cloud_context_application_context_services

Cloud Security: Offers a set of primitives to build secure applications and services using a declarative configurable model. Using Cloud Security, you can quickly create systems that implement common security features, such as single sign on, token relay and token exchange.

http://projects.spring.io/spring-cloud/spring-cloud.html#_spring_cloud_security

Cloud OAuth2: OAuth 2 An authorization framework that enables applications to obtain limited access to user accounts on an HTTP service, such as Facebook, GitHub, and Google. Cloud Oauth2. Cloud OAuth2 provides support for OAuth2 authorization server using the Spring Cloud.

http://stytex.de/blog/2016/02/01/spring-cloud-security-with-oauth2/

Cloud Task: Allows you to develop and run short lived microservices using Spring Cloud. You can run Cloud Task locally, in the cloud, or on Spring Cloud Data Flow. Batch applications is just one example use case of Cloud Task.

https://cloud.spring.io/spring-cloud-task/

CLOUD CONFIG

Config Server: Provides an HTTP, resource-based API for external configuration management of applications using name-value pairs, or equivalent YAML content. The server is easily embeddable in a Spring Boot application.

http://cloud.spring.io/spring-cloud-config/spring-cloud-config.html#_spring_cloud_config_server

Config Client: Binds to the Config Server and initialize Spring Environment with remote property sources. Config Client supports encryption and decryption of property values.

http://cloud.spring.io/spring-cloud-config/spring-cloud-config.html#_spring_cloud_config_client

Zookeeper Configuration: ZooKeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services. Spring Boot Zookeeper Configuration provides Apache Zookeeper integrations for Spring Boot

apps through autoconfiguration and bindings.

https://cloud.spring.io/spring-cloud-zookeeper/

Consul Configuration: Consul from HashiCorp is a highly available and distributed service discovery and key-value store designed with support for the modern data center. Consul Configuration provides integrations for Spring Boot applications with HashiCorp Consul through autoconfiguration and bindings.

https://cloud.spring.io/spring-cloud-consul/

CLOUD DISCOVERY

Eureka Server: Eureka Server provides a REST based service that is primarily used in the AWS cloud for locating services for the purpose of load balancing and failover of middle-tier servers. The Spring Boot Eureka Server starter provides support to Spring applications to standup a Eureka Service Registry that other applications can talk to.

https://spring.io/guides/gs/service-registration-and-discovery/#initial

Eureka Discovery: Eureka Discovery is a client-side service discovery that allows services to find and communicate with each other without hardcoding hostname and port. With Netflix Eureka, each client is able to simultaneously act as a server. Spring Boot Eureka Discovery allows Spring applications to interactively query Eureka Server given a service ID.

https://spring.io/guides/gs/service-registration-and-discovery/#initial

Zookeeper Discovery: ZooKeeper, a software project of the Apache Software Foundation is a distributed, open-source coordination service for distributed applications. Distributed applications can use ZooKeeper to implement higher level services for synchronization, configuration maintenance, and groups and naming. Spring Cloud Zookeeper Discovery provides Apache Zookeeper integrations for Spring Boot apps through autoconfiguration and bindings.

https://cloud.spring.io/spring-cloud-zookeeper/

Cloud Foundry Discovery: Cloud Foundry is an open source, multi cloud application platform as a service (PaaS) governed by the Cloud Foundry Foundation. Spring Cloud Foundry Discovery provides Apache Zookeeper integrations for Spring Boot apps through autoconfiguration and bindings.

https://docs.cloudfoundry.org/buildpacks/java/gsg-spring.html

Consul Discovery: Consul from HashiCorp is a highly available and distributed service discovery and key-value store designed with support for the modern data center. Consul Discovery provides integrations for Spring Boot applications for discovering Consul services.

https://cloud.spring.io/spring-cloud-consul/

CLOUD ROUTING

Zuul: Zuul is a JVM based router and server side load balancer by Netflix. Spring Cloud Zuul provides an embedded Zuul proxy to ease

the development of a very common use case where a UI application wants to proxy calls to one or more back end services.

https://bushkarl.gitbooks.io/spring-cloud/content/spring_cloud_netflix/router_and_filter_zuul.html

Ribbon: Ribbon is a client side load balancer which gives you a lot of control over the behaviour of HTTP and TCP clients Client Side Load Balancing (Ribbon) With Spring Boot Ribbon, you can build Spring Boot microservice application to provide client-side load balancing in calls to another microservice.

https://spring.io/guides/gs/client-side-load-balancing/

Feign: Feign is a Java to Http client binder. Feign aims to connect your code to HTTP API with minimal overhead and code. With Spring Boot Feign, you can create rest clients for all of your services, with minimal configuration and code.

https://bushkarl.gitbooks.io/spring-cloud/content/spring_cloud_netflix/declarative_rest_client_feign.html

CLOUD CIRCUIT BREAKER

Hystrix: Hystrix is a latency and fault tolerance library designed to isolate points of access to remote systems, services and 3rd party libraries. Spring Boot Hysterix allow a Spring microservice to continue operating when a related service fails.

https://spring.io/guides/gs/circuit-breaker/

Hystrix Dashboard: Provides a dashboard that displays the health of each circuit breaker from the set of metrics gathers about each Hystrix command. Hysterix Dashboard is itself another Spring Boot application.

http://cloud.spring.io/spring-cloud-netflix/spring-cloud-netflix.html

Turbine: Brings in an application that aggregates all of the relevant Hysterix stream endpoints into a combined Turbine stream for use in the Hystrix Dashboard. Note that Hystrix stream provides information on a single application, while Turbine provides a way to aggregate this information across all installations of an application in a cluster.

http://cloud.spring.io/spring-cloud-netflix/spring-cloud-netflix.html

Turbine Stream: Provides circuit breaker metric aggregation. You can configure Turbine Stream for a cluster and make it available at a URL. The Turbine Stream figures out the instances of the cluster using Eureka, source of the Hystrix stream from each instance, and aggregate them.

http://techblog.netflix.com/2012/12/hystrix-dashboard-and-turbine.html

CLOUD TRACING

Sleuth: Distributed tracing solution for Spring Cloud. Spring Cloud Sleuth borrows heavily from Dapper, Zipkin and HTrace to automatically instrument interactions with external systems in a manner that remains transparent to users. Spring CLoud Sleuth can capture data in logs, or by sending it to a remote collector service.

https://cloud.spring.io/spring-cloud-sleuth/

Sleuth Stream: In Spring Cloud Sleuth, a Span is the basic unit of work. For example, sending an RPC is a new span. Similarly sending a response to an RPC is another Span. With Sleuth Stream, you can accumulate and send span data over Spring Cloud Stream.

https://cloud.spring.io/spring-cloud-sleuth/spring-cloud-sleuth.html#_span_data_as_messages

Zipkin Server: A Spring Boot application, packaged as an executable jar where Span storage and collectors are configurable. By default, Zipkin server listens on port 9411, consumes Span data over HTTP, and uses in-memory Span storage.

https://cloud.spring.io/spring-cloud-sleuth/spring-cloud-sleuth.html#_terminology

Zipkin UI: The UI module of the Zipkin server that provides a single-page application to get a Zipkin service that accepts Spans and provide visualization to run queries.

https://github.com/openzipkin/zipkin/tree/master/zipkin-ui

Zipkin Stream: Consumes Span data in messages from Spring Cloud Sleuth Stream and writes them to a supported Zipkin store, such as in-memory, MySQL, Cassandra, and Elasticsearch.

https://spring.io/blog/2016/02/15/distributed-tracing-with-spring-cloud-sleuth-and-spring-cloud-zipkin

Zipkin Client: Enables distributed tracing with an existing Zipkin installation.

Cloud Messaging

Cloud Bus AMQP: Advanced Message Queuing Protocol (AMQP) is an open standard for passing business messages between applications or organizations. Cloud Bus AMQP links nodes of a distributed system with the AMQP message broker.

https://projects.spring.io/spring-amqp/

Stream Rabbit: RabbitMQ is open source message broker software that implements the Advanced Message Queuing Protocol (AMQP). Stream Rabbit allows creating Spring messaging microservices with RabbitMQ.

https://spring.io/guides/gs/messaging-rabbitmq/

Cloud Bus Kafka: Kafka is a fault-tolerant publish-subscribe messaging system. Cloud Bus Kafka provides a simple control bus with Kafka. You only need to enable the bus by adding this dependency and Spring Cloud takes care of the rest, provided that the Kafka broker is available and configured.

http://cloud.spring.io/spring-cloud-static/spring-cloud-bus/1.2.1.RELEASE/#_quick_start

Stream Kafka: Kafka is a fault-tolerant publish-subscribe messaging system. Stream Kafka allows creating Spring messaging microservices with Kafka.

https://cloud.spring.io/spring-cloud-stream/

Cloud AWS

AWS Core: Amazon Web Services (AWS) provides key infrastructure services for business through its cloud computing platform. AWS Core provides Spring Integration adapters for the various services provided by the AWS SDK for Java.

http://cloud.spring.io/spring-cloud-aws/spring-cloud-aws.html

AWS JDBC: Enables application developers to re-use their JDBC technology of choice and access the Amazon Relational Database Service with a declarative configuration.

http://cloud.spring.io/spring-cloud-aws/spring-cloud-aws.html#_data_access_with_jdbc

AWS Messaging: Provides Amazon Simple Queue Service (SQS) and Amazon Simple Notification Service (SNS) integration that simplifies the publication and consumption of messages over SQS or SNS.

http://cloud.spring.io/spring-cloud-aws/spring-cloud-aws.html#_messaging

Cloud Data Flow

Local Data Flow Server: Spring Cloud Data Flow is a cloud-native programming and operating model for composable data microservices on a structured platform. Local Data Flow Server provides a server implementation that runs locally.

https://cloud.spring.io/spring-cloud-dataflow/

Data Flow Shell: A client for the Data Flow Server. The shell allows you to perform the DSL command needed to interact with the server.

https://cloud.spring.io/spring-cloud-dataflow/

Cloud Cluster

Cluster Redis: Enables building "cluster" features, such as leadership election, consistent storage of cluster state, global locks, and one-time tokens with Redis,

http://docs.spring.io/spring-data/data-redis/docs/current/reference/html/#cluster

Cluster Zookeeper: Enables building features, such as leadership election, consistent storage of cluster state, global locks, and one-time tokens with Zookeeper ensemble.

https://zookeeper.apache.org/doc/r3.3.2/zookeeperAdmin.html#sc_zkMulitServerSetup

Cluster Hazelcast: Enables building "cluster" features, such as leadership election, consistent storage of cluster state, global locks, and one-time tokens with Hazelcat,

http://docs.hazelcast.org/docs/2.4/manual/html/ch14.html

Cluster Etcd: Enables building "cluster" features, such as leadership

election, consistent storage of cluster state, global locks, and one-time tokens with Etcd.

https://coreos.com/etcd/docs/latest/v2/clustering.html

Cloud Contract

Cloud Contract Verifier: Supports Consumer Driven Contracts and service schemas in Spring applications, covering a range of options for writing tests, publishing them as assets, asserting that a contract is kept by producers and consumers, for HTTP and message-based interactions.

https://cloud.spring.io/spring-cloud-contract/spring-cloud-contract.html

Cloud Contract Stub Runner: Stub Runner for HTTP/Messaging based communication

Cloud Contract WireMock: WireMock is a simulator, also known as service virtualization tool or a mock server for HTTP-based APIs. Cloud Contract WireMock enables using WireMock with different servers by using the "ambient" server embedded in a Spring Boot application.

https://cloud.spring.io/spring-cloud-contract/spring-cloud-contract.html#_spring_cloud_contract_wiremock

Pivotal Cloud Foundry

Config Client (PCF): Enables using a Spring Boot application as a client for a Config Server instance on Pivotal Cloud Foundry.

https://docs.pivotal.io/spring-cloud-services/1-3/config-server/writing-client-applications.html

Service Registry (PCF): Provides Spring Boot applications with an implementation of the Service Discovery pattern to perform Eureka service discovery on Pivotal Cloud Foundry

http://docs.pivotal.io/spring-cloud-services/1-3/service-registry/

Circuit Breaker (PCF): Pprovides Spring Boot applications with an implementation of the Circuit Breaker pattern as Hystrix circuit breaker on Pivotal Cloud Foundry

https://docs.pivotal.io/spring-cloud-services/1-3/circuit-breaker/

I/O

Batch: A batch framework designed to enable the development of batch applications vital for the daily operations of enterprise systems. Spring Batch provides reusable functions that are essential for batch processing – execution of a series of jobs that involves processing large volumes of records, including logging/tracing, transaction and resource management, and job processing statistics.

http://projects.spring.io/spring-batch/

Integration: A framework that provides out-of-the-box implementation of Enterprise Integration Patterns for building enterprise integration solutions. Spring Integration enables lightweight messaging within Spring-based applications and supports integration with external systems via declarative adapters.

https://projects.spring.io/spring-integration/

Activiti: Activiti is an enterprise Business Process Management (BPM) solution that takes as input a process definition comprised of human tasks and service calls and execute those in a certain order. Activiti exposes API's to start, manage and query data about process instances for that definition.

Spring Activiti provides a convention-over-configuration approach to integrate Spring application with the Activiti's process engine.

https://spring.io/blog/2015/03/08/getting-started-with-activiti-and-spring-boot

Apache Camel: Apache Camel is an open-source integration framework based on known Enterprise Integration Patterns. Spring Boot Apache Camel provides auto-configuration for Apache Camel.

http://camel.apache.org/spring-boot-example.html

JMS (ActiveMQ): Apache ActiveMQ is an open source message broker written in Java. Spring Boot JMS (ActiveMQ) enables publishing and subscribing to messages using ActiveMQ.

https://spring.io/guides/gs/messaging-jms/

JMS (Artemis): Artemis is an asynchronous messaging system for loosely coupled heterogeneous systems. With JMS (Artemis), Spring Boot auto-configures a ConnectionFactory when it detects that Artemis is available on the classpath.

http://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-messaging.html

JMS (HornetQ): HornetQ is an open source asynchronous Message-oriented middleware (MOM). Spring Boot JMS (HornetQ) enables using the Java Message Service API in a Spring Boot application with HornetQ. HornetQ, is supported in between Spring Boot 1.1.0.RELEASE and Spring Boot 1.4.0.RC1. Post Spring Boot 1.4.0.RC1, HornetQ is deprecated in favor of Artemis.

http://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-messaging.html

AMQP: Applies core Spring concepts to the development of AMQP-based messaging solutions. It provides a "template" as a high-level abstraction for sending and receiving messages.

https://projects.spring.io/spring-amqp/

Mail: Provides a higher level of abstraction for sending electronic mail. Spring Mail shields a user from the specifics of underlying mailing system and is responsible for a low level resource handling on behalf of the client.

http://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-email.html

LDAP: Simplifies LDAP operations, based on the pattern of Spring's JdbcTemplate. Spring Boot LDAP requires Spring Boot 1.5.0.RC1 or higher.

http://projects.spring.io/spring-ldap/

Ops

Actuator: Adds several production grade services to your application for monitoring and managing your applications. Spring Boot Actuator exposes information via 'endpoints'. Production ready features to help you monitor and manage your application.

https://springframework.guru/chuck-norris-for-spring-boot-actuator/

Actuator Docs: Provides API documentation for the Actuator endpoints

https://github.com/spring-projects/spring-boot/tree/master/spring-boot-actuator-docs

Remote Shell: Supports an integrated Java shell called CRaSH that you can use to ssh or telnet into your running application.

http://docs.spring.io/spring-boot/docs/current/reference/html/production-ready-remote-shell.html

Experimental

Reactive Web: An experimental project that provides a simple way to try the new Web Reactive support in Spring Framework 5.0. Reactive programming is about non-blocking, event-driven applications that scale with a small number of threads. Reactive Web supports development with Apache Tomcat and requires Spring Boot 2.0.0.BUILD-SNAPSHOT or higher.

https://spring.io/blog/2016/02/09/reactive-spring