Scaling data services with pivotal gemfire pdf

The following apps provide additional help with Spring Boot development with the support of Apache Geode or Pivotal GemFire. Most often ask the question: What spring data for the Apache Geode/Pivotal GemFire annotation I can or should use in the development of Apache Geode or Pivotal GemFire applications with Spring Boot? To answer this question, we should start by reviewing the full collection of available spring data for Apache Geode/Pivotal GemFire (SDG). These annotations are presented in the package org.springframework.data.gemfire.config.annotation. Most of the relevant annotations start at @Enable... with the exception of basic annotations: @ClientCacheApplication, @PeerCacheApplication and @CacheServerApplication. What's more, the Spring Download for Apache Geode/Pivotal GemFire (SBDG) is based on a SDG-based configuration model to implement automatic configuration and use basic Spring Boot concepts such as the configuration convention to create GemFire/Geode applications with Boot Spring securely, quickly and easily. SDG provides this annotation-based configuration model to give app developers a choice when building Spring applications using Apache Geode or Pivotal GemFire. SDG makes no assumptions about what app developers are trying to do, and not quickly at any time the configuration is ambiguous, giving users immediate feedback. Second, SDG annotations were designed to launch app developers quickly and reliably. SDG achieves this by applying reasonable defaults, so app developers don't need to know or even explore all the complex configuration details and tools GemFire/Geode provides to perform simple tasks, such as prototyping. So SDG is all about choice, and SBDG is all about convention. Together, this framework provides app developers with convenience and reliability to move quickly and easily. To learn more about the motivation behind the SDG-based configuration model, check out the background. SBDG currently provides an automatic configuration for the following functions: ClientCache Cash-abstraction Continuous execution of the request function - Implementation of the PDX GemfireTemplate Spring Security Data Repository (customer/server Auth sSL) Spring session Technically, this means that the following SDG annotations are not required to use the above features: @ClientCacheApplication @EnableGemfireCaching (or using Spring Framework's @EnableContinuousQueries @EnableGemfireFunctionExecutions @EnableGemfireFunctions @EnableLogging @EnablePdx @EnableGemfireRepositories @EnableSecurity @EnableSsl @EnableGemFireHttpSession @EnableCaching) SBDG automatically adjusts these features for you, then higher annotations are not strictly required. Typically, you only announce one of your myk annotations when you want to override the Spring Boot conventions expressed in automatic configuration and adjust function behavior. In this section, we cover a few examples to make the behavior of override more obvious. By default, SBDG provides you with a copy of ClientCache. Technically, SBDG achieves this by annotating the auto configuration class with @ClientCacheApplication, internally. Under the convention, we assume that most app developers will develop Spring Boot applications using Apache Geode or Pivotal GemFire as client applications in the topology of the GemFire/Geode client. This is especially true as users migrate their applications to a manageable environment such as Pivotal CloudFoundry (PCF) using Pivotal Cloud Cache (PCC). However, users can override their default settings and declare their Spring applications as actual peer-to-peer cluster members. @SpringBootApplication @CacheServerApplication MySpringBootPeerPeerCache ServerPrime ... By announcing the @CacheServerApplication annotation, you're actually canceling SBDG by default. Therefore, SBDG will not provide a copy of ClientCache because you have informed SBDG exactly about what you want, i.e. a peer-to-peer instance of Cache hosting a built-in CacheServer that allows customer connections. However, you might ask: Well, how do you set up a copy of ClientCache when developing client applications without direct announcement @ClientCacheApplication annotations then? First, you can fully configure a ClientCache instance by explicitly announcing the @ClientCacheApplication in the Spring Boot configuration and install certain attributes if necessary. However, you should be aware that by explicitly announcing this annotation, or any of the other auto-configured default annotations, then you take on all the responsibility that comes with it, since you are actually re-warming the automatic configuration. One example of this is the security we touch on more below. The most ideal way to configure any feature is to use known and documented properties listed in the Spring Boot.properties app (convention), or with a configurator. For more information, see the Background Guide. Like @ClientCacheApplication, annotation @EnableSecurity is not strictly required unless you want to override and customize by default. Outside of a controlled environment, the only security configuration required is to provide the username and password. You do this using the famous and SDG document username/password properties in the Spring Boot application.properties as well: safety properties in the restlessness of Envionment Envionment spring.data.gemfire.security.password-Secret You don't need to explicitly declare an annotation @EnableSecurity just to specify a security configuration (such as a username/password). Inside a managed environment such as pivotal CloudFoundry (PCF) using Pivotal Cloud Cache (PCC), SBDG can introspect the environment and set up Security (Auth) completely without having to specify any configuration, usernames/passwords or otherwise. This is partly because PCF provides security information in the VCAP environment when the application is deployed on PCF and tied to services (such as PCC). So, in short, you don't need to explicitly announce @EnableSecurity annotations (or @ClientCacheApplication for that matter). However, if you directly announce either @ClientCacheApplication and/or @EnableSecurity annotations, guess what, now you are responsible for this configuration and the automatic SBDG configuration no longer applies. While direct ad @EnableSecurity makes more sense when it redefines the automatic configuration of SBDG Security, direct announcement of the @ClientCacheApplication annotation is likely to make less sense about its impact on the security configuration. This is entirely due to the internal GemFire/Geode, which in some cases, as security, is not even able to completely shield users from the nuances of the GemFire/Geode configuration. Both Auth and SSL must be configured before creating a cache instance (whether it's ClientCache or peer cache, it doesn't matter). Technically, this is due to the fact that security is enabled/configured during the construction of the cache. And the cache pulls the configuration out of the JVM properties that need to be installed before the cache is built. Structuring the exact order of the automatic configuration classes provided by SBDG when classes are triggered is no small feat. So it's no surprise that security's automatic configuration classes in SBDG should be run before the ClientCache auto-configuration class, so a copy of ClientCache can't auto-authenticate properly in the PCC when @ClientCacheApplication is explicitly announced without any help (i.e. you should also explicitly announce the annotation @EnableSecurity in this case, since you override the automatic cache configuration, and, well, not the security. Again, this is due to how security (Auth) and SSL meta data should be provided by GemFire/Geode. For more information, see the Background Guide. Most of the time, many of the other auto-configured annotations for C, Features, PDX, Repository, and so on, don't need to ever be announced explicitly. Many of these features are automatically activated by having SBDG or other libraries (such as the Spring Session) on the classpath, included based on other annotations applied to spring spring beans Let's take a look at a few examples. When using SBDG, it is rare, if all, to declare directly either @EnableCaching the spring framework @EnableGemfireCaching or @EnableGemfireCaching of a specific SDG annotation. SBDG automatically caching and customizes SDG GemfireCacheManager for you. You just need to focus on which components of the application service are suitable for caching: CustomerService @Service Service - @Autowired private customerRepository customerRepository; @Cacheable (CustomersByName) findBy (String name) - customerRepository.findByName (name); Of course, you need to create GemFire/Geode Regions that support caches claimed in application service components (e.g. CustomersByName) using Spring Caching annotations (such as @Cacheable), or as an alternative, JSR-107, JCache (e.g., '@CacheResult). You can do this by defining each region explicitly, or more conveniently, you can simply use: Setting caches (regions) @SpringBootApplication @EnableCachingDefinedRegions class Of Apps @EnableCachingDefinedRegions... For more information, see the Background Guide. Rarely, if all, you need to explicitly declare SDG @EnableContinuousQueries annotations. Instead, you should focus on identifying application requests and worry less about plumbing. Determining requests for public class @Component TemperatureMonitor expands AbstractTemperatureEventPublisher - @ContinuousQuery (name - BoilingTemperatureMonitor, request - SELECT - FROM /TemperatureReadings WHERE temperature.measurement zgt; 212.0) public void of boilingTemperatureReadings (CqEvent event) temperatureReading -'gt; new BoilingTemperatureEvent (it's, temperatureReading)); - @ContinuousQuery (title - FreezingTemperatureMonitor, query - SELECT - FROM /TemperatureReadings WHERE temperature.measurement qlt; 32.0) public void of freezingTemperatureReadings (event CqEvent) temperatureReading - qgt; new FreezingTemperatureEvent (it, temperatureReading); Of course, GemFire/Geode C' applies only to customers. For more information, see the Background Guide. Rarely, if all, it is necessary to expressly state either @EnableGemfireFunctionExecutions, @EnableGemfireFunctions annotations. SBDG provides an automatic configuration for both performance and performance. You just need to determine the implementation: The implementation function of the @Component class GemFireFunctions - @GemfireFunction Object exampleFunction (object arg)... And then determine the execution: @OnRegion (region - Example) interface GemFireFunctionExecutions - Example of objectFunction (Object arg); SBDG will automatically find, set up and register the feature (POJOs) in GemFire/Geode as proper functions, and create proxy proxies for interfaces that can then be entered into application service components to call registered features without the need for direct announcement of favorable annotations. Application and execution implementation functions (interfaces) should simply exist below the @SpringBootApplication annotated core class. More than a detailed guide can be seen in the zgt.. Rarely, if anything, you need to directly announce @EnablePdx annotation, as SBDG automatically sets up PDX by default. SBDG automatically adjusts SDG MappingPdxSerializer as the default pdxSerializer. It's easy to set up a PDX configuration by installing the appropriate properties (PDX search) in the Spring Boot app. For more information, see the Background Guide. Rarely, if all, you need to directly declare an annotation @EnableGemfireRepositories, as SBDG automatically adjusts the Spring Data (SD) repository by default. You just need to identify your repositories and get cranked up: CustomerRepository Customer Repository Interface Expands CrudRepository'lt; Customer, Long----FoundByName (String name); SBDG finds repository interfaces defined in your application, proxy telephony, and registers them as beans in the spring applicationContext. Repositories can be entered into other components of the application service. Sometimes it's convenient to use @EnableEntityDefinedRegions with SD repositories to identify the entities used by the app and identify the regions used by the SD repository infrastructure to keep the entity state. The abstract @EnableEntityDefinedRegions optional, provided for convenience and free for @EnableGemfireRepositories annotation. For more information, see the Background Guide. Most of the other annotations in SDG focus on specific application issues or allow for certain GemFire/Geode features rather than being a necessity. @EnableAutoRegionLookup @EnableBeanFactoryLocator @EnableCacheServer (s) @EnableCachingDefinedRegions @EnableClusterConfiguration @EnableCompression @EnableDiskStore (s) @EnableEntityDefinedRegions @EnableEviction @EnableExpiration @EnableGemFireAsLastResource @EnableHttpService @EnableIndexing @EnableOffHeap @EnableLocator @EnableManager @EnableMemcachedServer @EnablePool (s) @EnableRedisServer @EnableStatistics @UseGemFireProperties None of these annotations are required, and none of them are set up by SBDG automatically. They are simply at the disposal of app developers, if and when necessary. It also means that none of these annotations is in conflict with any automatic SBDG configuration. Finally, it's important to understand where it ends and SBDG begins. It all starts with the automatic configuration provided by SBDG out of the box. If the function is zlt;/Customer, qgt; zgt; In the automatic SBDG configuration, you are responsible for turning on and setting up the function appropriately as your application is needed (such as @EnableRedisServer). In other cases, you can also explicitly declare a free annotation (such as @EnableEntityDefinedRegions) for convenience, as there is no convention or opinion provided by SBDG out of the box. In all other cases, it comes down to understanding how GemFire/Geode works under the hood. While we do our best to protect users from as many details as possible, it is not appropriate or practical to address all issues, such as cache creation and security. I hope that this section provided some relief and clarity. The following two reference sections cover documented and well-known properties recognized and processed by Spring Data for Apache Geode/Pivotal GemFire (SDG), and the Spring Session for Apache Geode/Pivotal GemFire (SSDG). These properties can be used in Spring Boot application.properties files or as JVM properties to customize different aspects or incorporate individual Apache Geode or Pivotal GemFire features into the Spring app. Combined with the power of Spring Boot, magical things begin to happen. All of the following properties have a spring.data.gemfire console. For example, to install a cache copying property on the reading, use spring.data.gemfire.cache.copy-on-read in Spring Boot application.properties. Table 22. spring.data.gemfire. SpringBasedCacheClientApplication ClientCacheApplication.name Comma-delimited locators list of locator endpoints, formatted as: locator1-port1 ,... ,locatorN-portN. PeerCacheApplication.locators-use-be-factory-locator-incorporate-SDG BeanFactoryLocator when mixing Spring configurations with GemFire/Geode native configuration (such as cache.xml) and you'll want to customize GemFire objects announced in cache.xml with spring. ClientCacheApplication.useBeanFactoryLocator Table 24. spring.data.gemfire.' ClientCache Properties Title Default description from cache.client.durable-client-id Is only used for customers in the client/server installation. If set, it means that the customer is durable and identifies the customer. The ID is used by servers to recover any messages interrupted by downtime. ClientCacheApplication.durableClientId cache.client.durable-client-timeout Is only used for customers in a client/server setting. The number of seconds a client can stay disconnected from his server and have the server continue to accumulate solid events for him. 300 ClientCacheApplication.durableClientTimeout cache.client.keep-alive Setting up whether the server should keep strong customer queues alive during the timeout period. False Таблица 25. spring.data.gemfire. spring.data.gemfire. Description of the default name from cache.peer.enable-auto-reconnect Setting whether a member (locators and servers) will try to restore and restore the cache settings after it has been forced to leave the cluster by the event section of the network or otherwise avoided by other members. false PeerCacheApplication.enableAutoReconnect cache.peer.lock-lease adjusts the length, in a matter of seconds, distributed lock lease agreements obtained by this cache. 120 PeerCacheApplication.lockLease cache.peer.lock-timeout Sets up the number of seconds that a cache operation will wait to get a distributed lease lock. 60 PeerCacheApplication.lockTimeout cache.peer.message-sync-interval adjusts the frequency (in seconds) from which the main cache server sends a message to all secondary cache server nodes to remove events that have already been sent out of the queue. 1 PeerCacheApplication.messageSyncInterval cache.peer.search-timeout 300 PeerCacheApplication.searchTimeout cache.peer.use-cluster-configuration whether this GemFire cache member site is configured for a meta-data configuration from the cluster configuration cluster service. PeerCacheApplication.useClusterConfiguration CacheServer can be further targeted at specific cacheServer instances using the name of the CacheServer bean option, defined in the context of the Spring app. For example: spring.data.gemfire.cache.server. (.bind-address)... DiskStore properties can be further targeted at specific DiskStores with the help of DiskStore.name. For example, you can specify the location of directory files for a specific one called DiskStore using: spring.data.gemfire.store.store.example.directory.location/path/to/geode/disk-stores/Example/ Director's location and file size diskStore can be further divided into multiple locations and sizes using array syntax, as in: spring.data.gemfire.disk.store.Example.directory.0.location/path/to/geode/disk-stores/Example/one spring.data.gemfire.disk.store.example.directory.0.0.. size-4096000 spring.data.gemfire.disk.disk. store. Example.directory.1'.location/path/to/geode/disk-stores/Example/two spring.data.gemfire.disk.store.example.directory-1'.size-8192000 Both names and index arrays are optional and you can use any combination of name and index array. Nameless properties apply to all DiskStores. Without array indices, all DiskStore files will be stored at a specified location and limited to a certain size. Table 29. spring.data.gemfire.' Properties entity Description of the default name from entities.base-packages Comma-delimited list of package names showing starting points to scan the entity. EnableEntityDefinedRegions.basePackages Table 30. spring.data.gemfire. By default, locator.host sets up an IP address or host name for the NIC system, to which the built-in Locator will be required to listen to connections. EnableLocator.host locator.port sets up a network port to which the built-in Locator will listen to connections. 10334 EnableLocator.port Table 31. spring.data.gemfire. EnableLogging.logLevel logging.log-disk-space-limit customizes the amount of disk space that allows you to store log files. EnableLogging.logDiskSpaceLimit logging.log-file sets the name of the way the log file used to register messages. EnableLogging.logFile logging.log-file-size sets the maximum size of the log file before the folding log file. IncludeLogging.logFileSize Table 32. spring.data.gemfire. Property Management Title Default description from management.use-http Customizes does use the HTTP protocol to communicate with GemFire/Geode Manager. False EnableClusterConfiguration.useHttp management.http.host sets up the IP address or host name of GemFire/Geode Manager, HTTP. EnableClusterConfiguration.host management.http.port sets up a port used by GEMFire/Geode manager's service to listen to connections. 7070 EnableClusterConfiguration.port Table 33. The property manager's default name description from manager.access-file sets up an access management list (ACL) file used by the manager to restrict access to JMX MBeans by customers. EnableManager.accessFile manager.bind-address sets up the IP address or host name of the NIC system used by the manager to link and listen to JMX client connections. EnableManager.bindAddress manager.hostname-for-clients sets up the owner's name given to JMX customers to ask Locator about the manager's location. EnableManager.hostNameForClients manager.password-file By default JMX Manager will allow customers without credentials to connect. If this property is set up in the file name, only customers who connect to accounts that match the entries in that file are allowed. EnableManager.passwordFile manager.port sets up the port used by the JMX Customer Manager. 1099 EnableManager.port manager.start sets up whether to run the Manager service during the run. False EnableManager.start manager.update-rate adjusts the speed in milliseconds with which this participant will push updates for any JMX managers. 2000 EnableManager.updateRate Table 34. spring.data.gemfire.» PDX Properties Default name description from pdx.disk-store-name customizes the name DiskStore is used to store meta-data type PDX on the drive when PDX is persistent. EnablePdx.diskStoreName pdx.ignore-unread-fields ignores fields that were unread during the desirization. False EnablePdx.ignoreUnreadFields pdx.persistent Adjusts whether PDX retains a type of meta-data on the drive. False EnablePdx.persistent pdx.read-serialized Whether the region record returns as PdxInstance or deserialized back to the subject form to read. False EnablePdx.readSerialized pdx.serialize-bean-name-sets the name of custom spring beans by implementing org.apache.geode.pdx.PdxSerializer. EnablePdx.serializerBeanName Table 35. spring.data.gemfire.' Pool Properties Title Description from pool.free-connection-time-out Sets time out used to purchase free connection from the pool. 10000 EnablePool.freeConnectionTimeout pool.idle-timeout Adjusts the amount of time the connection can wait until the connection expires (and closes). 5000 EnablePool.idleTimeout pool.load-conditioning interval for how often the pool will check whether the connection to that server should be relocated to another server to improve the load balance. 300000 EnablePool.loadConditioningInterval pool.locators Comma-delimited list of Locator endpoints in format: locator1',... ,locatorN'portN'EnablePool.locators pool.max-sets up the maximum number of customers to server connections that will create a pool. EnablePool.maxConnections pool.min-connections customizes a minimum number of customers to server connections that will support the pool. 1 EnablePool.minConnections pool.multi-user-authentication Customizes whether the pool created can be used by multiple authenticated users. False EnablePool.multiUserAuthentication pool.ping-interval sets up how often ping servers are to make sure they are still alive. 10000 EnablePool.pingInterval pool.pr with one hop-enabled Setting up whether to perform one-hop operations between the customer and the servers. When this is true, the customer is aware of the location of sections on servers that host regions with DataPolicy.PARTITION. True EnablePool.prSingleHopEnabled pool.read-timeout sets up a number of milliseconds to wait for a server response before you exit the operation and try another server (if any). 10000 EnablePool.readTimeout pool.ready-for-events Sets up whether to signal to the server that the client is ready to receive events. ClientCacheApplication.readyForEvents pool.retry-attempts Sets up the number of times to repeat the request after a timeout/exclusion. The EnablePool.retryAttempts pool.server-group sets up a group to which all the servers to which the pool is connected belong. IncludePool.serverGroup pool.servers Comma-delimited list of CacheServer endpoints in format: server1'port1',... ,serverN'portN' EnablePool.servers pool.buffer-socket-size The size of the socket buffer for each connection made in all pools. 32768 32768 pool.statistical-interval adjusts how often to send customer statistics to the server. EnablePool.statisticInterval pool.subscription-ack-interval sets up a millisecond interval to wait before submitting confirmations to CacheServer for events received from server subscriptions. 100 EnablePool.subscriptionAckInterval pool.subscription with support Customizes whether the created pool will have a server to the customer subscription enabled. The false attribute of EnablePool.subscriptionEnabled pool.subscription-message-tracking-timeout adjusts the messageTrackingTimeout attribute, which is a period of time before life, in milliseconds, for subscription events that the client received from the server. 900,000 EnablePool.subscriptionMessageTrackingTimeout pool.subscription-redundancy adjusts the redundancy level for all server-to-client subscriptions. EnablePool.subsriptionRedundancy pool.thread-local-connections-sets up a local flow connections policy for all pools. False EnablePool.threadLocalConnections Table 36. spring.data.gemfire.' Security Properties Default Name Description from security.username sets up the username used to authenticate from servers. EnableSecurity.securityUsername security.password sets up the user password used to authenticate from servers. Enable Security.securityPassword security.properties-file customizes the system's path name to a property file containing security credentials. EnableAuth.propertiesFile security.client.accessor X EnableAuth.clientAccessor security.client.accessor-post-processor Callback, which should be called at the postoperative stage, that is, when the operation is completed on the server, but before the result is sent to the customer. IncludeAuth.clientAccessorPostProcessor security.client.authentication-initializer Include Security.clientAuthentiationInitializer security.client.authenticator.authenticator.authenticator.com.au, returning the Authenticator object used by a member of the cluster (Locator, Server) to verify the account of the connected customer. EnableAuth.clientAfenticoctor security.client.diffie-hellman-algorithm is used for authentication. To secure the transfer of sensitive credentials, such as passwords, you can encrypt credentials using the Diffie-Hellman key-sharing algorithm. Do this by installing the security-client-dhalgo system property on customers in the name of a valid, symmetrical key cipher supported by JDK. EnableAuth.clientDiffieHellmanAlgorithm security.log.file sets the path name to the log file used for security log messages. EnableAuth.securityLogFile security.log.level sets up a log level for security log messages. security.manager.class-name Include Security.securityManagerClassName security.peer.authentication-initializer EnableSecurity.peerAuthenticationInitializer security.peer.authenticator The static creation method returns an authentication object that is used peer-to-peer to verify the connection site's credentials. EnableAuth.peerAuthenticator security.peer.verify-member-timeout sets up time out in milliseconds used by a peer-to-peer team to verify the membership of an unknown authenticated peer request for a secure connection. EnableAuth.peerVerifyMemberTimeout security.post-processor.class-name sets the class name by implementing the interface org.apache.geode.security.PostProcessor, which can be used to change the returned results of the Region's operations. EnableSecurity.securityPostProcessorClassName security.shiro.ini-resource-path customizes the Apache Geode system property, citing the location of the Apache Shiro INI file, which adjusts the Apache Shiro security framework to ensure Apache Geode security. IncludeSecurity.shiroIniResourcePath Table 37. spring.data.gemfire.' SSL Properties Default Name Description from security.ssl.certificate.alias.cluster sets up a pseudonym for a saved SSL certificate used by the cluster to provide communication. EnableSsl.componentCertificateAliases security.ssl.certificate.alias.alias sets up the default alias on the stored SSL certificate used to provide communications throughout the GemFire/Geode system. EnableSsl.defaultCertificateAlias security.ssl.certificate.alias.gateway adjusts the alias to the stored SSL certificate used by WAN Gateway senders/receivers to ensure communication security. EnableSsl.componentCertificateAliases security.ssl.certificate.alias.jmx sets up a pseudonym for a saved SSL certificate used by JVMM MBeanServer and JMX customer servicer to ensure communication security. EnableSsl.componentCertificateAliases security.ssl.certificate.alias.locator sets up a pseudonym for a saved SSL certificate used by Locator to ensure communication security. EnableSsl.componentCertificateAliases security.ssl.certificate.alias.server sets up a pseudonym for a saved SSL certificate used by customers and servers to ensure communication security. EnableSsl.componentCertificateAliases security.ssl.certificate.alias.web sets up a pseudonym for a saved SSL certificate used by the built-in HTTP server for communication security (HTTPS). EnableSsl.componentCertificateAlias security.ssl.ciphers Comma-divided list of SSL ciphers or whatever. IncludeSsl.ciphers security.ssl.components Comma-delimited list of GemFire/Geode components (e.g. WAN) that will be configured for Ssl. EnableSsl.components security.ssl.keystore customizes the system's path name to a Java KeyStore file that stores certificates for SSL. Ssl. security.ssl.keystore.password adjusts the password used to access the Java KeyStore file. EnableSsl.keystorePassword security.ssl.keystore.type adjusts the password used to access the Java KeyStore file (e.g. JKS). EnableSsl.keystoreType security.ssl.protocols-comma-divided-list of SSL protocols or whatever. Turn onSl.protocols security.ssl.require-authentication Customizes whether two-year authentication is required. EnableSsl.requireAuthentication security.ssl.truststore customizes the system's path name to the trust store (file Java KeyStore), retaining certificates for SSL. EnableSsl.truststore security.ssl.truststore.password adjusts the password used to access the trust store (file Java KeyStore). EnableSl.truststorePassword security.ssl.truststore.type adjusts the password used to access the trust store (file photo Java KeyStore; for example JKS). EnableSl.truststoreType security.ssl.web-requires authentication To set up whether bilateral authentication of HTTP is required. False EnableSsl.webRequireAuntia Table 38. spring.data.gemfire. Service Properties Default Name Description from service.http.bind-address sets up the IP address or host name of the NIC system used by the built-in HTTP server to bind and listen to http (S) connections. EnableHttpService.bindAddress service.http.http.port sets up the port used by the built-in HTTP server to listen to HTTP (S) connections. 7070 EnableHttpService.port service.http.ssl-require-authentication Customizes whether two-part http authentication is required. False EnableHttpService.sslRequireAuthentication service.http.dev-rest-api-start-set-up whether to start a web service Developer REST API. You need a full Apache Geode or Pivotal GemFire installation, and you have to install $GEODE environment. False EnableHttpService.startDeveloperRestApi service.memcached.port sets up the port of the built-in memcached server (service). 11211 EnableMemcachedServer.port.memcached.protocol.com ASCII EnableMemcachedServer.protocol service.redis.bind-address sets up the IP address or host name of the NIC system used by the built-in Redis server to link listening to connections. EnableRedis.bindAddress service.redis.port sets up the port used by the built-in Redis server to listen to connections. Although it's not advisable to use Apache Geode properties directly in Spring apps, SBDG won't stop you from doing so. A full link to the specific Apache Geode properties can be found here. Apache Geode (and Pivotal GemFire) are very strict about properties that may be listed in the gemfire.properties file. Spring properties cannot be mixed with gemfire properties. If you how to disable the automatic configuration of any function provided by Spring Boot for Apache Geode/Pivotal GemFire, then you can specify the automatic configuration class in excludes the attribute of the @SpringBootApplication annotation, as follows: Disable the Automatic configuration of PDX @SpringBootApplication (excluded - PdxSerializationAutoConfiguration.class) public class MySpringBootApplication - public class MySpringBootApplication - public class you can disable more than 1 class of automatic configuration at the same time by specifying each class in the exception attribute with the array syntax, as follows: Disable the AUTOMATIC CONFIGURATION of PDX and SSL @SpringBootApplication (exclude - PdxSerializationAuto Configuration.class, SslAutoConfiguration.class The current set of spring-loading automatic configuration classes for Apache Geode and Pivotal GemFire include: CacheNameAutoConfiguration CachingProviderAutoConfiguration ClientCacheAutoConfiguration ClientSecurityAutoConfiguration ContinuousTimeAutofiguration FunctionExecutionAutoConfiguration GemFire LoggingAutoConfiguration PdxSerializationAutoConfiguration PeerSecurityAutoConfiguration RegionTemplateAutoConfiguration RepositoriesAutoConfiguration SpringSessionAutoConfiguration SpringSessionAutoPropertiesSergity SslAutoConfi3guration First, understand, that Pivotal GemFire is replaced by Pivotal Cloud Cache (PCC). Thus, all references to Pivotal GemFire (i.e. gemfire) also imply for Pivotal Cloud Cache (i.e. cloud) as well. When it comes to supporting Spring, whether you're developing open source software (OSS) Apache Geode or developing for Pivotal Cloud Cache, Spring has you covered. At a strategic level, this means, from open source software (such as Apache Geode) to commercial (e.g., a key cloud cache) from non-management environments (e.g. standalone, externally managed) to managed environments (such as a key platform) with little or no need for code or configuration changes. It just works! You can also go back and move your Spring Boot apps from Pivotal Platform to commercial software, Pivotal Cloud Cache, and return to Open Source Apache Geode, operating in a standalone, externally controlled environment. SBDG won't (always) lock you up! It's your choice! Technically, this means to go from Apache Geode to Pivotal Cloud Cache, you only need to change SBDG's dependence on: Maven POM with spring download for Apache Geode zlt;dependency'gt; &lt;artifactId&gt;весна-geode-стартер&lt;/artifactId&gt; &lt;version&gt;1.1.10.RELEASE&lt;/version&gt; &lt;/dependency&gt; Gradle построить файл с весенней загрузкой для Apache Geode Geode compilation 'org.springframework.geode:spring-geode-starter:1.1.10.RELEASE' Maven POM with spring download for Pivotal GemFire The Gradl to build a spring download file for GemFire's core dependencies is a compilation of 'org.springframework.geode:spring-gemfire-starter:1.1.1.10.RELEASE' follow these instructions in pivotal Gem. To go back, a simple change of spring-gemfire-starter back to the spring-geode-starter. Done! The Spring Boot automatic configuration and convention on the configuration approach tries to determine the run time environment to cope with infrastructure logistics, so you don't have to. This is true inside or out of control It should just work without any code or configuration changes, and if it is not, for whatever reason, then we will work to fix it, due to any feature differences between Pivotal Cloud Cache that cannot be performed with Apache Geode by itself. To go back, a simple change of spring-gemfire-starter back to the spring-geode-starter. The Spring Boot Automatic Configuration and Convention on Configuration approach tries to determine the timing environment so that we can provide users with a consistent and reliable experience without all the hassle and challenges of switching environments. Switching environments is especially common when you switch Spring Boot apps from DEV to TEST, to STAGING, and finally to PRODUCTION. Of course, it will almost always be easier to run Apache Geode as a managed service inside Pivotal Platform with Key Cloud Cache than to manage an outside-controlled Apache Geode cluster, especially if your case requires maximum performance and high availability. We strongly recommend this approach whenever and wherever possible, but it is still your choice. As described in the Building ClientCache app, you can set up and run a small cluster of Apache Geode or Pivotal GemFire from inside your IDE with Spring Boot. This is extremely useful during development, as it allows you to manually twist, test, and debug applications quickly and easily. The spring download for Apache Geode/Pivotal GemFire includes this class: Spring Boot app class, used to customize and enhance the security of the Apache Geode/Pivotal GemFire @SpringBootApplication @CacheServerApplication (name - SpringBootApacheGeodeCacheServerApplication) @SuppressWarnings (unused) public class SpringBootApacheGeCacheServerApplication - public static emptiness core (String) Args) - new SpringApplicationBuilder .web (WebApplicationType.NONE) .build() .run(args); .run(args); @Configuration @UseLocators @Profile (cluster) static ClusterConfiguration class - @Configuration @EnableLocator @EnableManager (beginning - true) @Profile (!cluster) static LonerConfiguration class - this class is the right Spring Boot application that can be used to set up and download multiple Apache Geode or Pivotal GemFire servers and combine them together to form a small cluster, simply changing the time it takes. Initially you want to start one, a main server with a built-in Locator and a service manager. The Locator service allows cluster members to find each other and allows new members to try to join the cluster as a peer group. In addition, Locator also allows customers to connect to cluster servers. When a cache client pool is configured to use locators, the pool can reasonably route data requests directly to the server that receives the data (also as single-currency access), especially when the data is separated/shards on the cluster servers. Locator pools include support for balancing load connections and handling automatic sleep-for in case of failed connections, among other things. Manager lets you connect to this server with Gfsh (Apache Geode tool and Pivotal GemFire shell). To run our main server, create a launch configuration in your IDE for springBootApacheGeodeServerApplication with the following recommended JRE command line options: Server Profile Configuration 1 -server-ea-Dspring.profiles.active. You should see a similar exit: Server 1 launch /Library/Java/JavaVirtualMachines/jdk1.8.0_152.jdk/Contents/Home/bin/java-server-ea-Dspring.profiles.active-javaagent:/applications/applications/Intelli J IDEA 17 CE.app/Contents/lib/idea_rt.jar-62866:/Applications/IntelliJ IDEA 17 CE.app/Contents/bin-Dfile.encoding-UTF-8 -classpath org.springframework.geode.docs.example.app.server.SpringBootApacheGeodeServerApplication SLF4J: I couldn't download the class org.slf4j.impl.StaticLoggerBinder. SLF4J: Default to non-operation (NOP) SLF4J implementation registrar: See for more details. . ____ _ __ _ _ /\\ / ___'_ __ _ _(_)_ __ __ _ \ \ \ \ (( )___ | '_ | '_| | '_ \/ _` | \ \ \ \ \\/ ___)| |_)| | | | | || (_| |)))) ' |____| .__|_| |_|_| |___, | / / / / : : Spring download :: (v2.0.3.RELEASE) Info 2018/06/24 21:42:22 8.183 PDT zlt Starting with SpringBootApacheGeodeCacheServerApperApplication on jblum-mbpro-2.local with PID 41795 (/Users/jblum/piv jblum/spring-boot-data-geode/spring-geode-docs/build/classes/main, jblum developed in /Users/jblum/pivdev/spring-download-data-geode/spring-geode-doc Info 2018/06/24 21:42:28.186 PDT Retreat to default profiles: default Info 2018/06/24 21:42:28.278 PDT qlt'gt'tid'0x1 Refreshing org.springemail protected6fa51cd4: launch date Sun June 24 21:42:28 PDT 2018; The root of the contextual hierarchy warn 2018/06/24 21:42:28.962 PDT zlt't'tid'0x1 @Bean method PdxConfiguration.pdxDiskStoreAwareBeanFactoryPostProcessor is not static and returns the object assigned to the BeanFaanFayPostOror interface. This will result in an inability to process annotations such as @Autowired, @Resource @PostConstruct, and @PostConstruct in declaring the method @Configuration class. Add a static modifier to this method to avoid these container lifecycle problems; see @Bean Javadoc for more information. Info 2018/06/24 21:42:30.036 PDT -- license to the Apache Software Foundation (ASF) under one or more licensing agreements. For more information on copyright ownership, please visit the NOTICE file distributed by this work. ASF licenses this file to you under the Apache license, Version 2.0 (License); You can't use this file unless it's in compliance with the License. You can obtain a copy of the license in Unless applicable law or agreed in writing, the software distributed under the License is covered by AS IS BASIS, UN WARRANTIES OR CONDITIONS OF ANY KIND, or express or imply. You can see the License for a specific language that regulates permits and restrictions under the License. -- Build-Date: 2017-09-16 07:20:46 -0700 Build-Id: abaker 0 Build-Java-Version: Построенная платформа: Mac OS X 10.12.3 x86_64 Продукт-Имя: Apache Geode Продукт-Версия: 1.2.1 Источник-Дата: 2017-09-08 11:57:38 -0700 Источник-Репозиторий: релиз/1.2.1 Источник-пересмотр:&lt;/main&gt; &lt;/main&gt; &lt;/main&gt; &lt;/main&gt; &lt;/main&gt; &lt;/main&gt; Родная версия: родной код недоступен Запуск на: /10.0.0.121, 8 cpu (s), x86_64 Mac OS X 10.10.5 Коммуникационная версия: 65 Process ID: 41795 Пользователь: jblum Current dir: /Users/jblum/pivdev/spring-boot-data-geode/spring-geode-docs/build Home dir: /Users/jblum Command Line Parameters: -ea-Dspring.profiles. :/Приложения/IntelliJ IDEA 17 CE.app/Contents/lib/idea_rt.jar-62866:/Applications/IntelliJ IDEA 17 CE.app/Contents/bin -Dfile.encoding-UTF-8 Class Path: /Библиотека/Java/JavaVirtualMachines/jdk1.8.0_152.jdk/Contents/Home/jre/charsets.jars.... Путь библиотеки: /Пользователи/jblum/Библиотека/Java/Расширения /Библиотека/Java/Расширения /Сеть/Библиотека/Java/Расширения /Система/Библиотека/Java/Расширения /usr/lib/java . Свойства системы: PID No 41795 ... «Info 2018/06/24 21:42:30.045 PDT &lt;main&gt;tid'0x1» Конфигурация стартапа: «GemFire Свойства, определяемые с api » отключить-авто-reconnect-true jmx-manager-true jmx-manager-port-1099 jmx-manager-start'true j mmx-менеджер-обновление-ставка-2000 журнал-уровень'config mcast-port'0 имя ВеснаBootApacheGeodeCacheCacheServerApplication start-locator'localhost10334 использовать-кластер-конфигурацияложные GemFire Свойства с использованием значений по умолчанию ack-severe-alert-порог Информация 2018/06/24 21:42:30.090 PDT tid'0x1 Начало расположения одноранговой связи для &lt;main&gt;распределительного локатора на localhost/127.0.0.1 (инфо) 2018/06/24 21:42:30.093 PDT &lt;main&gt;tid'0x1 Начиная распределение Локатор на localhost/127.0.0.1 (инфо 201) 8/06/24 21:42:30.094 PDT &lt;main&gt;tid'0x1 Локатор был создан в Sun Jun 24 21:42:30 PDT 2018 (инфо 20)18/06/24 21:42:30.094 PDT &lt;main&gt;tid'0x1 Слушая на порт 10334 связаны по адресу localhost/127.0.0.1 ... Информация 2018/06/24 21:42:30.685 PDT &lt;main&gt;tid'0x1» Инициализирующий регион _monitoringRegion_10.0.0.121&lt;v0&gt;1024 (инфо) 2018/06/24 21:42:30.688 PDT &lt;main&gt;tid'0x1 Инициализация региона _monitoringRegion_10.0.0.121&lt;v0&gt;1024 завершена ... [info 2018/06/24 21:42:31.570 PDT &lt;main&gt;tid=0x1] CacheServer Configuration: port=40404 max-connections=800 max-threads=0 notify-by-subscription=true socket-buffer-size=32768 maximum-time-between-pings=60000 maximum-message-count=230000 message-time-to-live=180 eviction-policy=none capacity=1 overflow directory=. groups=[] loadProbe=ConnectionCountProbe loadPollInterval=5000 tcpNoDelay=true [info 2018/06/24 21:42:31.588 PDT &lt;main&gt;tid=0x1] Started SpringBootApacheGeodeCacheServerApplication in 3.77 seconds (JVM running for 5.429) You can now connect to this server using Gfsh: Connect with Gfsh $ echo $GEMFIRE /Users/jblum/pivdev/apache-geode-1.2.1 jblum-mbpro-2:lab jblum$ jblum-mbpro-2:lab jblum$ gfsh __________________________ __ / _____/ ______/ ______/ /____/ / / / __/ /___ /_____ / _____ / / / __/ / ____/ _____/ / / / / /_______/_/ /______/_/ /_/ 1.2.1 Monitor and Manage Apache gfsh&gt;connect Connecting to Locator at [host=localhost, port=10334] .. Подключение к менеджеру в&lt;/main&gt; &lt;/main&gt; &lt;/v0&gt; &lt;/main&gt; &lt;/v0&gt; &lt;/main&gt; &lt;/main&gt; &lt;/main&gt; &lt;/main&gt; &lt;/main&gt; &lt;/main&gt; &lt;/main&gt; port No.1099. Успешно подключен к: «хозяин» 10.0.0.121, порт »1099» gfsh&gt;list Члены Имя Id -- -- ВеснаBootApacheGeodeCache СерверАппликация (ru) 10.0.0.121 (ВеснаBootApacheGeodeCacheServerApplication:41795)&lt;ec&gt;&lt;v0&gt;:1024 gfsh&gt;отписи члена --имя --имя'SpringBootApacheGeodeCacheCacheСейтерАппликация Имя : SpringBootApacheGeodeCacheServerApplication Id : 10.0.0.121 (ВеснаBootApacheGeodeCacheServerApplication:41795)&lt;ec&gt;&lt;v0&gt;:1024 Хозяин : 10.0.0.0.121 Регионы : PID : 41795 Группы : Подержанные кучи : 184M Макс Хип : 3641M Рабочий Dir : /Пользователи / jblum/pivdev/весна-загрузка-данные-geode/весна-geode-docs / построить файл журнала : /Пользователи / jblum/pivdev /весна-загрузка-данные-geode/весна-geode-docs/построить Локаторы : localhost '10334 Кэш Сервер Информационный сервер Bind : Сервер Порт : 40404 Запуск : истинные подключения клиентов : 0 Теперь, давайте начнем некоторые дополнительные серверы для масштабирования нашего кластера. To do this, you just need to change the name of the members that we will add to our cluster as peers. Apache Geode and Pivotal GemFire require members of the cluster to be named and each member of the cluster named as unique. In addition, since we are launching several instances of our SpringBootApacheGeodeServerServlication class, which also embeds a cacheServer instance that allows cache customers to connect, we should be careful to vary our ports used by the built-in services. Fortunately, we don't need to run another built-in Locator or Manager service (in this case we only need 1), so we can switch profiles from the unsure to use the spring cluster profile, which includes a different configuration (cluster configuration class) to connect another server as a peer in the cluster, which currently has only 1 member, as shown in the Gfsh command output list above. To add another server, place the participant's name and CacheServer port on another number with the following launch profile configuration: You run a 2 server configuration for the server-ea-Dspring.profiles.active-clustered-Dspring.data.gemfire.name-ServerTwo-Dspring.data.gemfire.cache.server.port-41414 Notification that we have clearly activated the Spring Profile cluster, which allows the configuration presented in the nested cluster configuration of the class when disabling the LonerConfiguration class. This ClusteredConfiguration class is also annotated @UseLocators, which installs the GemFire/Geode locator property on local hosting. By default, it assumes that the Locator process/service works on locahost by listening to the default port of Locator 10334. Of course, you can set up the ultimate Locators, if your locators are operating elsewhere on your network, using the locator attribute @UseLocators annotations. In production environments, you can often run multiple locators as a ltlt. a separate process. The launch of several locators provides redundancy in the case of locator flushing. If all Locator processes on your network fail, don't worry, your cluster won't go down. This simply means that no other members will be able to join the cluster, allowing you to scale the cluster, nor will customers be able to connect. Just restart the locators if that happens. In addition, we spring.data.gemfire.name ServerTwo property by adjusting our member's name when he joins the cluster as a peer. Finally, we installed spring.data.gemfire.cache.server.port at 41414 to vary the CacheServer port used by ServerTwo. The default port of CacheServer is 40404. If we hadn't installed this property before ServerTwo started, we would have been in java.net.BindException. spring.data.gemfire.name and spring.data.gemfire.cache.server.port are well-known properties used by SDG to dynamically customize GemFire/Geode using the Spring Boot.properties file or Java System properties. These properties can be found in Javadoc annotations in the SDG-based configuration model. For example, spring.data.gemfire.cache.server.port is documented here. Most SDG annotations include appropriate properties that can be identified in application.properties and are described in more detail here. После запуска нашего второго сервера, ServerTwo, мы должны увидеть аналогичный выход на командной строке, и в Gfsh, когда мы перечисляем членов и описываем участника снова: выход Gfsh после запуска сервера 2 --- gfsh&gt;list членов Имя Id -- -- ВеснаBootApacheGeodeCache СерверАппликация (ru) 10.0.0.121 (ВеснаBootApacheGeodeCacheServerApplication:41795)&lt;ec&gt;&lt;v0&gt;:1024 ServerTwo 10.0.0.121 (ServerTwo:41933)&lt;v1&gt;:1025 gfsh&gt; Автор - name-ServerTwo Имя : ServerTwo Id : 10.0.0.121 (ServerTwo:41933)&lt;v1&gt;:1025 Хост : 10.0.0.121 Регионы : PID : 41933 Группы : Использованная куча : 165M Max Heap : 3641M Рабочий Dir : /Пользователи/jblum/pivdev/весна-загрузка-данные-geode/весна-geode-docs/построить файл журнала : /Пользователи/jblum/pivdev/весна-boot-data-geode/весна-geode-docs/локаторы сборки : localhost '10334 Кэш Сервер Информационный сервер Bind : Сервер Порт : 41414 Запуск : истинные подключения клиентов : 0 --- Когда список членов, мы видим , ServerTwo, and when we describe ServerTwo, we see that its cacheServer port is properly installed at 41414. If we add another server, ServerThree using the following launch configuration: Add server 3 to our cluster-server-ea-Dspring.profiles.active-clustered-Dspring.data.gemfire.name-ServerThree-Dspring.data.gemfire.cache.server.port-42424 Again, we'll see a similar на командной линии и в Gfsh: Gfsh выход после запуска сервера 3 gfsh&gt;лист членов Имя Id&lt;/v1&gt; &lt;/v1&gt; &lt;/v0&gt;&lt;/ec&gt; &lt;/v0&gt;&lt;/ec&gt; | --- SpringBootApacheGeodeCache ServerApplication (SpringBootApacheGeodeCacheServerApplication:41795)wo:41933):v1'gt;1025 ServerThree 10.0.0.121 (ServerThree:41965) 0.121 (ServerThree:419 65) spring-download-data-geode/spring-geode-docs/build-log file:/users/jblum/pivdev/spring-loading-data-geode/spring-geode-docs/build-locators: localhost '10334 Cash Server Information Server Bind : Server Port: Port Server: 42424 You just started a small Cluster Apache Geode/Pivotal GemFire with 3 members using Spring Boot from inside your IDE. It's pretty easy to build and run a spring download, Apache Geode/Pivotal GemFire, a ClientCache app that connects to this cluster. Just turn on and use the spring download for Apache Geode/Pivotal GemFire, ;-). In fact, the entire test package in the spring download for Apache Geode and Pivotal GemFire is based on this project. All spring projects integrated with either Apache Geode or Pivotal GemFire will use this new test base for all their testing needs, making this new test base for Apache Geode and Pivotal GemFire a proven and reliable solution for all your testing needs for Apache Geode/Pivotal GemFire applications when using Spring. Later, this guide will include and dedicate a whole chapter on testing. The final source of the truth on how best to use the spring download for Apache Geode and Pivotal GemFire (or Pivotal Cloud Cache (PCC)) is to refer to samples. Check out the key cloud cache (PCC), The Pizza Store, the Spring Boot app for example using Spring Boot for Pivotal GemFire (SBDG) in the ClientCache app, woven into the PCC. In addition, you can refer to the App Temperature, a Spring Download that implements the temperature sensor and monitoring, the Internet of Things (IOT) example. The example uses SBDG to demonstrate Apache Geode C, implementation/execution function, and positions Apache Geode as a caching provider in spring cache abstraction. This is a working, complex and complete example, and is highly recommended as a good starting point for real-world uses. Another example is the example of downloading from the Contact Application Reference (RI) for spring data for Apache Geode and Pivotal GemFire (SDG). Page 2 For transmission data must be serialized between customers and servers, or when data is distributed and replicated between peers in a cluster. In this case, the data in question is the state of the session. At any time, the session is a session. or access to the topology of the client/server, the state of the session is sent by wires. Typically, the Spring Boot app with the spring session enabled will be a server client (s) in the Apache Geode node cluster. On the server side, the session state may be spread across multiple servers (data nodes) in the data replication cluster and ensure that the session state is highly accessible. Using Apache Geode, data can be separated or shards, and redundancy levels can be specified. When data is distributed for replication, it must also be serialized to transmit the session status between the nodes of the node in the cluster. Outside the box, Apache Geode supports Java serialization. Java Serialization has many advantages, such as processing cycles on the object graph, or universal support for any java-written application. However, Java Serialization is very verbose and is not the most effective over-the-wire format. Thus, Apache Geode provides its own serialization framework for serializing Java types: As mentioned above, Apache Geode provides 2 additional serialization frameworks: data serialization and PDX serialization. Data serialization is a very efficient format (i.e. fast and compact) with little overhead compared to Java Serialization. It supports Delta Distribution by sending only bits of data that have actually changed, as opposed to sending an entire object, which certainly reduces the amount of data sent over the network in addition to reducing IO when data is stored or overflowed onto the drive. However, the serialization of the data means that the processor is fine any time the data is transmitted over the wire, or stored/overwhelmed and accessible from the disk, as the receiving end performs desirialization. In fact, anytime Delta Spread is used, the object must be deserialized at the receiving end in order to apply the delta. Apache Geode uses deltas, referring to the method at the facility that implements the interface org.apache.geode.Delta. Obviously, you can't call a method on a serialized object. PDX, on the other hand, which advocates portable data sharing, retains the form in which the data was sent. For example, if a customer sends data to a PDX-style server, the server will store the data as PDX serial bytes and store it in the cache region for which the data access operation was directed. In addition, PDX, as the name suggests, is portable, which means that it allows Java and Native Language customers, such as customers C, C, and C, to work on the same data set. PDX even allows you to execute O'L queries on serialized bytes without causing objects to be demerized in the first place to evaluate the predicate of the request and execute the request. This can be achieved, how Apache Geode supports a registry type containing a type of meta-data for objects that are serialized stored in Apache Geode using PDX. However, portability does come with a price tag, having a little more overhead than serializing data. However, PDX is much more efficient and flexible than Java Serialization, which stores a type of meta-data in serialized bytes of an object, rather than in a separate type registry, as in the case of Apache Geode when using PDX. PDX does not support Deltas. Technically, a pdX serial object can be used in Delta Propagation by implementing the interface of org.apache.geode.Delta, and only the delta will be shipped, even in the context of PDX. But then, the PDX serialized object must be deserialized to apply the delta. Remember that the method is called for applying delta, which defeats the goal of using PDX in the first place. In developing indigenous clients (such as C) who manage data in a storage name cluster, or even when mixing indigenous customers with Java customers, there will generally be no related Java types provided on the trajectory of servers in the cluster. With PDX, there is no need to provide Java types on the activity trajectory, and many users who are just developing and using Indigenous customers to access data stored in the storage name will not provide any types of Java for the respective C/C/C types. Apache Geode also supports JSON serialized in/from PDX. In this case, it's very likely that Java types won't be provided on the server class trajectory, because Apache Geode can use many different languages (such as JavaScript, Python, Ruby) that support JSON. However, even with PDX in play, users should take care not to call the PDX serialized object on the servers in the cluster to be deserialized. For example, consider O'L's request for a Java-type object serially published as PDX... @Region (People) of the Person class - private LocalDate birthDate; The line's private name Public int getAge () / // No explicit field/property age in a person // Age is simply realized in terms of the birthDate field - afterwards, if O'L's request triggers a method on a person's object, for example: SELECT - FROM /People p.age zgt;' 21 Then it will cause the PDX Person to be de-realized, as age is not a human field, but rather a method containing calculations based on another field of the person (i.e. birthDate). Similarly, calling any java.lang.Object method in O'L's query, like Object.toString, will also cause desirialization. Apache Geode provides a read-action configuration, so any Region.get (key) cache operations that are potentially called inside the function do not result in the de-image of PDX serial objects. But nothing will prevent O'L's ill-conceived request to cause desirialization, so be Apache Geode can support all 3 serialization formats at the same time. For example, an app domain model may contain that implement the java.io.Serialiable interface, and you can use a combination of data serialization frameworks with PDX. When using Java Serialization with serialization and PDX, it is generally preferable to use a serialization strategy 1. Unlike Java Serialization, data serialization and PDX serialization do not process object graph cycles. For more information on Apache Geode serialization mechanics, click here. Earlier, the Apache Geode spring session only supported the Apache Geode data serialization format. The main motivation for this was to use the Delta Propagation Apache Geode feature, as the session state can be arbitrarily large. However, as the spring session for Apache Geode 2.0 is held, PDX is also supported and is now the new default serialization option. The default was changed to PDX in the spring session for Apache Geode 2.0 primarily because PDX is the most widely used and requested format by users. PDX is by far the most flexible format, so much so that you don't even need a spring session for Apache Geode or any of its transit dependencies on classpath servers in the cluster to use the spring session with Apache Geode. In fact, with PDX, you don't even need to put the types of domain applications stored in (HTTP) Sessions on classpath servers either. In fact, using PDX serialization, Apache Geode does not require related Java types to be present on the class trajectory of servers. As long as there is no deserilization on the cluster servers, you're safe. The @EnableGemFireHttpSession abstract introduces a new sessionSerializer BeanName, which the user can use to customize the name of the beans announced and registered in the Spring container, implementing the desired serialization strategy. The serialization strategy is used by Spring Session for Apache Geode to serialize the session state. Outside the box, the Spring Session for Apache Geode provides 2 serialization strategies: 1 for PDX and 1 for serializing data. It automatically registers both bean serialization strategies in the Spring container. However, only 1 of these strategies are actually used while running, PDX! The two beans registered in the spring container, the data serialization and PDX, are called SessionDataSerializer and SessionPdxSerializer, respectively. By default, sessionSerializerBeanName is configured on SessionPdxSerializer, as if the user were annotated for their spring download, The spring session included an app configuration class with: @SpringBootApplication @EnableGemFireHttpSession (sessionSerializerBeanName - SessionPdxSerializer) class MySpringSessionApplication . . . . It's a simple matter to change the serialization strategy for data serialization, not by installing the sessionSerializerBeanName attribute SessionDataSerializer as @SpringBootApplication @EnableGemFireHttpSession (sessionSerializerBeanName - SessionDataSerializer) class MySpringSessionApplication . . . . Because these 2 values are so common, the spring session for Apache Geode provides constants for every value in the GemFireHttpsessionConfiguration class: GemFireHttpSessionConfiguration.SESSION_PDX_SERIALIZER_BEAN_NAME and GemFireHttpSessionConfiguration.SESSION_DATA_SERIALIZER_BEAN_NAME. So you can clearly set up PDX, as follows: import org.springframework.session.data.geode.config.annotation.web.http.GemFireHttpsesionConfiguration; @SpringBootApplication @EnableGemFireHttpSession (sessionSeriserBeanNaim and GemFireHttpSessionConfiguration.SESSION_PDX_SERIALIZER_BEAN_NAME) Class MySpringSession Application . . . . With 1 attribute and 2 provided definitions of beans out of the box, you can specify what structure of serialization you want to use with your spring download, the spring session enabled the app with the support of Apache Geode. To abstract the details of the Apache Geode sequence and PDX serialization framework, the Apache Geode Spring Session provides its own serialization structure (facade), wrapping the Apache Geode serialization framework. The serialization API exists in the org.springframework.session.data.gemfire.serialization. The main interface of this API is org.springframework.session.data.gemfire.serialization.SessionSerializer. The interface is defined as: Spring SessionSerializer SessionSerializer interface, in, out-of-the-post serialize (T-session, OUT); T deserialize (IN in); boolean canSerialize (Type of zlt;?? Class;) boolean canSerialize (Object obj) -/ Calls Object.getClass () in a zero safe way, followed by calls and returns canSerialize (:Class) - Basically, the interface allows you to serialize and desirialize the object of a spring session. In and OUT type settings and related method arguments of these types provide a reference to the objects responsible for writing session to the byt stream or reading the Session from the bytes stream. The actual arguments will be specific to the types, based on the configured Apache Geode serialization strategy. For example, if you use the PDX Apache Geode series framework, IN and OUT are copies of org.apache.geode.pdx.PdxReader and org.apache.geode.pdx.PdxWriter, respectively. When the Apache Geode data serialization structure is configured, IN and OUT are copies of java.io.DataInput and java.io.DataOutput, respectively. These arguments are provided in the implementation of the SessionSerializer framework and, as mentioned earlier, is based on Apache Geode's core serialization strategy. Essentially, despite the fact that the spring session for Apache Geode,lt;'t,'gt; provides a facade around the Apache Geode serialization framework, under the hood of Apache Apache still expects one of these serialization structures to be used to serialize data in/from Apache Geode. So what is the purpose of the SessionSerializer interface really serving then? In fact, it allows the user to customize which aspects of the session state actually gets serialized and stored in Apache Geode. App developers can provide their own custom, application-specific SessionSerializer implementation, register it as beans in the Spring container, and then customize it to use a spring session for Apache Geode to serialize the session state, as follows: @EnableGemFireHttpSession (sessionSerializerBeanName - MyCustomSessionSerializer) class MySpringsionDataGemFireApplication - @Bean (MyCustomSessionSerializer) SessionSerializer zlt;session,'gt;zth; The spring session for Apache Geode provides help when a user wants to implement a custom SessionSerializer that fits into one of Apache Geode's serialization structures. If the user simply implements the interface org.springframework.session.data.gemfire.serialization.SessionSerializer directly, Without expanding from one of the spring sessions to provided by Apache Geode abstract basic classes related to 1 of Apache

Geode's serialization framework, the spring session for Apache Geode will wrap up the custom implementation of sessionSerializer in the case of org.springframework.session.data.gemfire.serialization.pdx.support.PdxSerizer The spring session for Apache Geode is careful not to stomp on any existing PdxSerializer implementation that the user may have already registered with Apache Geode by some other means. Indeed, there are several different, subject to the implementation of Apache Geode's interface org.apache.geode.pdx.PdxSerializer: This is achieved by obtaining any currently registered instance of PdxSerializer in the cache and compiling it with pdxSerializerSesionSerializerAdapter packaging of the user of the SessionSerizer application. The pdxSerializer composite implementation is provided by the spring session for the Apache Geode class org.springframework.session.data.gemfire.pdx.support.ComposablePdxSerializer class when objects are stored in Apache Geode as PDX. If no other PdxSerializer is currently registered in the Apache Geode cache, the adapter is simply registered. Of course, you can force the basic Apache Geode Serialization strategy used with the custom implementation of SessionSerializer by doing one of the following: custom sessionSerializer implementations can implement Apache Geode at org.apache.geode.pdx.PdxSerializer or for convenience, extend the spring session for Apache Geode's Class and Spring Session for Apache Geode will register custom SessionSerializer as a PdxSerializer with Apache Geode. Custom Implementations SessionSerializer can expand Apache Geode to org.apache.geode.DataSerializer class, or for convenience, extend the spring session for Apache Geode at org.springframework.session.data.gemfire.serialization.data.abstractDataSerializableSesionSerializer class and spring session for Apache Geode will register a custom Finally, the user can create a custom implementation SessionSerializer, as before, without specifying what structure of the Apache Geode series to use, because the custom implementation of SessionSeriaizer does not implement any interfaces serialization Apache Geode or expands from any spring session to provided Apache Geode abstract base classes, and until now it is registered in Apache Geode as DataSerializer, announcing an additional spring session for Apache Geode beans in the spring container type org.springframework.session.data.gemfire.serialization.data.support.DataSerializerSesionSerializerAdapter as it is ... Forcing a custom SessionSerializer to register as a DataSerializer in Apache Geode @EnableGemFireHttpSession (sessionSerializerBeanName - customSessionSerializer) app class - @Bean DataSerializerSesionSerializer ializer () - the return of new DataSerializerSesionSesion - @Bean SessionSerializer zlt;Session, registered as beans in a Spring container, any neutral user implementation of SessionSerializer will be considered by DataSerializer's Retailer in Apache Geode. Please feel free to skip this section if you set up and load Apache Geode servers into a cluster using Spring (Boot), as usually the information that follows will not be applied. Of course, it all depends on the stated dependencies and configuration of Spring. However, if you're using Gfsh to run servers in a cluster, be sure to read on. When using the Apache Geode DataSerialization framework, especially from the client in the serialization (HTTP) session to the cluster servers, you need to take care to set up Apache Geode servers in a cluster with appropriate dependencies. This is especially true when using delta, as explained in the previous section on data serialization. When you use data frameworking as a serialization strategy for serialization (HTTP) of session status from web application customers to servers, servers should be correctly configured with the spring session for the Apache Geode class types used to present the session (HTTP) and its contents. This means that including Spring Classpath. In addition, using DataSerialization may also require you to include JARs containing application domain classes that your web application uses and entered into (HTTP) Session as session attribute values, especially if: Your types implement the interface org.apache.geode.DataSerializable. Your types implement the interface org.apache.geode.Delta. You've registered org.apache.geode.DataSerializer, which defines and serializes types. Your types implement the java.io.Serializable interface. Of course, you should make sure that the types of application domain objects placed in the (HTTP) session are serialized in one form or another. However, you are not required to strictly use DataSerialization and you don't necessarily have to have the types of application domain objects on the classpath server if: your types implement org.apache.geode.pdx.PdxSerializable interface. Or you've registered org.apache.geode.pdx.PdxSerializer, which correctly identifies and serializes the types of application domain objects. Apache Geode will prioritize the definition of a serialization strategy for serializing objects: first, DataSerializable objects and/or any registered DataSerializers identified by objects for serialization. Then PdxSerializable objects and/or any registered pdxSerializer identifying objects for serialization. And finally, all types of java.io.Serializable. This also means that if a certain type of application domain object (e.g. A) implements java.io.Serializable, however, (user) PdxSerializer has been registered with Apache Geode, identifiable by the same type of application domain object (i.e. A), Apache Geode will use PDX to serialize A, not Java Serialization, in this case. This is especially useful since you can use DataSerialization to serialize the session object (HTTP) using Deltas and all the powerful DataSerialization features, but then use PDX to serialize the types of application domain objects, making it much easier to configure and/or related efforts. Now that we have a common understanding of why this support exists, how do you allow it? First, create an Apache Geode.xml cache, as follows: Apache Geode cache.xml?xml version?1.0 encoding'UTF-8?'lt;lt;cache xmlns' :xsi' xsi:schemalocation/ version zlt.initializer.com-class-name.org.springframework.session.data.gemfire.serialization.data.support.DataSerializableSessionSerializerInitializer Then run your servers in Gfsh with: Start a server with Gfsh gfsh. --cache-xml-file/path/to/cache.xml.xml.xml...- Настройка Настройка Geode server classpath with the appropriate dependencies is the tricky part, but generally, the following should work: CLASSPATH configuration set variable --name=REPO_HOME --value=${USER_HOME}/.m2/repository gfsh&gt; start server ... --classpath=\ ${REPO_HOME}/org/springframework/spring-core/5.1.18.RELEASE/spring-core-5.1.18.RELEASE.jar\ :${REPO_HOME}/org/springframework/spring-aop/5.1.18.RELEASE/spring-aop-5.1.18.RELEASE.jar\ :${REPO_HOME}/org/springframework/spring-beans/5.1.18.RELEASE/spring-beans-5.1.18.RELEASE.jar\ :${REPO_HOME}/org/springframework/spring-context/5.1.18.RELEASE/spring-context-5.1.18.RELEASE.jar\ :${REPO_HOME}/org/springframework/spring-context-support/5.1.18.RELEASE/spring-context-support-5.1.18.RELEASE.jar\ :${REPO_HOME}/org/springframework/spring-expression/5.1.18.RELEASE/spring-expression-5.1.18.RELEASE.jar\ :${REPO_HOME}/org/springframework/spring-jcl/5.1.18.RELEASE/spring-jcl-5.1.18.RELEASE.jar\ :${REPO_HOME}/org/springframework/spring-tx/5.1.18.RELEASE/spring-tx-5.1.18.RELEASE.jar\ :${REPO_HOME}/org/springframework/data/spring-data-commons/2.1.20.RELEASE/spring-data-commons-2.1.20.RELEASE.jar\ :${REPO_HOME}/org/springframework/data/spring-data-geode/2.1.20.RELEASE/spring-data-geode-2.1.20.RELEASE.jar\ :${REPO_HOME}/org/springframework/session/spring-session-core/2.1.13.RELEASE/spring-session-core-2.1.13.RELEASE.jar\ :$REPO_HOME/org/springframework/session/spring-session-data-geode/2.1.11.RELEASE/spring-session-data-geode-2.1.11.RELEASE.REPO_HOME jar. About what you might need to add a JAR domain object to the server class. To get a complete picture of how it works, see the sample. By default, any time the session changes (e.g., the last Unavailable time is updated to the current time), the session is considered a messy spring session for Apache Geode (SSDG). When you use the framework to serialize Apache Geode data, it's extremely useful and valuable to also use Apache Geode Delta Distribution capabilities. When using data serialization, SSDG also uses Delta Propagation to send only changes to the session status between the client and the server. This includes any session attributes that can be added, removed or updated. By default, anytime Session.setAttribute (name, value) is called, the session attribute is considered dirty and will be sent to the delta between the client and the server. This is true even if the application domain object has not been changed. There is usually never a reason to call Session.setAttribute (..) unless your object has been changed. However, if this can happen, and your objects are relatively large (with a complex hierarchy of objects), then you may want to consider either: Implementing the Delta interface in the application domain object model, while useful, is very invasive, or ... UI implementation of the SSDG org.springframework.session.data.gemfire.support.IsDirtyPredicate. Out of the box, SSDG provides 5 implementations of the IsDirtyPredicate strategy interface: Table 5. IsDirtyPredicate implementation Description the default class description IsDirtyPredicate.ALWAYS_DIRTY the New Session attribute is always considered dirty. IsDirtyPredicate.NEVER_DIRTY new Session attributes are never considered dirty. DeltaAwareDirtyPredicate New Session attributes are considered messy when the old value and new value are different if the new value type does not implement Delta or the Delta.hasDelta method returns correctly. Yes EqualsDirtyPredicate The new session attribute values are considered dirty iff the old value does not equal the new value defined by Object.equals (:Object). IdentityEqualsPredicate New Session attributes are considered dirty, the old value is not the same as the new value using an identification equivalent to the operator (i.e. oldValue! - newValue). As shown in the table above, DeltaAwareDirtyPredicate is the default implementation used by SSDG. DeltaAwareDirtyPredicate automatically takes into account the domain objects of the applications that implement the Apache Geode Delta interface. However, DeltaAwareDirtyPredicate works even when the application domain objects do not implement the Delta interface. SSDG will make the application domain object dirty anytime it is called Session.setAttribute (name, newValue), ensuring that the new value is not the same as the old value, or the new value does not implement the Delta interface. You can change the dirty implementation of SSDG, a definition strategy by simply announcing the beans in the Spring interface container IsDirtyPredicate type: Redefining SSDG default IsDirtyPredicate strategy @EnableGemFireHttpSession class ApplicationConfiguration - @Bean IsDirtyPredicate equals DirtyPredicate () return EqualSDirtyPredicate. The IsDirtyPredicate also provides AndThen (:IsDirtyPredicate) and orThen (:IsDirtyPredicate) to compile 2 or more IsDirtyPredicate implementations in composition to organize complex logic and rules to determine whether an application domain object is dirty or not. For example, you can make up as EqualsDirtyPredicate, and DeltaAwareDirtyPredicate via OR operator: Making EqualDirtyPredicate with DeltaAwareDirtyPredicate using a logical operator OR @EnableGemFireHttpSession class applicationConfiguration - @Bean IsDirtyPredicate equalsOrThenDeltaDirtyPredicate () - Return Of Accurate. In the... You can even implement your own, custom IsDirtyPredicates based on specific types of application domain objects: Type-specific IsDirtyPredicate application type object CustomerDirtyPredicate class implements IsDirtyPredicate - public bulean isDirty (Object oldCustomer, NewCustomer Object) - if (newCustomer instanceof Customer) / // Custom logic to determine whether the new Customer is dirty, the return is true; - AccountDirtyPredicate class implements IsDirtyPredicate - public boolean isDirty (Object oldAccount, Object newAccount) - if (newAccount instanceof Account) then combine CustomerDirtyPredicate with AccountDirtyPredicate and default predicate for return, as follows: Compiled and configured type-specific IsDirtyPredicates @EnableGemFireHttpSession Class ApplicationConfiguration - @Bean IsDirtyPredicate typeSpecificDirtyPredicate () - the return of the new CustomerDirtyPredicate () .andThen (new AccountDirtyPredicate ()).andThen (IsDirtyPredicate.ALWAYS_DIRTY); The combinations and possibilities are endless. Use caution when implementing custom IsDirtyPredicate strategies. If you misidentified that the application domain object isn't dirty when it's actually, then it won't be sent to the session delta from client to server. Inside, the Spring Session for Apache Geode supports 2 presentations (HTTP) sessions and session attributes. Each view is based on whether Apache Geode Deltas is supported or not. The spread of the Apache Geode delta is only included in the spring session for Apache Geode when using serialization of data for reasons previously discussed. In fact, the strategy is that if Apache Geode serialization is configured, Deltas is supported and used by DeltaCapableGemFireSession and DeltaCapableGemFireSessionAttributes. If the Apache Geode PDX Serialization is configured, Delta Propagation will be disabled, and GemFireSession and GemFireSessionAttributes submissions will be used. You can override these internal views used by Spring Session for Apache Geode, and for users to provide their own types related to the session. The only strict requirement is that the session should implement the main interface of the Spring Session org.springframework.session.session.Session. For example, let's say you want to define your own Session implementation. First, you determine the type of session. It's possible that the custom session type even encapsulates and handles session attributes without having to identify a separate type. The user-defined MySession session implementation class is org.springframework.session.session. Then you need to expand org.springframework.session.data.gemfire.GemFireationsSessionRepository and override the createSession method to create instances of the session implementation custom class. MySessionRepository Expands GemFireOperationsSsionRepository - @Override public session createSession - the return of the new MySession (); If you provide your own custom SessionSerializer implementation and Apache Geode PDX Serialization is configured, then you did. However, if you've configured Apache Geode Data Serialization, you should additionally provide the UI implementation of the SessionSerializer interface and either have it directly expand Apache Geode in the org.apache.geode.DataSerializer class, or extend the spring session for Apache Geode's org.springframework.session.data.gemfire.serialization.data.AbstractDataSerializableSessionSerializer class and override getSupportedClasses (): Class'lt;??-gt; method. Custom SessionSerializer for a custom session type mySessionSerializer expands AbstractDataSerializableSesionSerializer - @Override public class? getSupportedClasses (. . . . He said , he said that I. A- - I. . . . . Unfortunately, getSupportedClasses can't bring back the generic type of Spring Session interface org.springframework.session.session.Session. If this were possible, we could avoid the obvious need to override the dataSerializer user implementation method. But, Apache Geode's data serialization framework can only fit into exact class types because it is incorrectly and internally stored and refers to a class type by name, which then requires the user to override and implement the getSupportedClasses () method. Method. scaling data services with pivotal gemfire pdf. scaling data services with pivotal gemfire pdf download. scaling data services with pivotal gemfire github

normal_5f874452a8e94.pdf
normal_5f873136acca0.pdf
normal_5f876147a59fb.pdf
normal_5f87c4c8939c5.pdf
badlands 2500 winch installation instructions
euro truck simulator 2 indir sağlam indir
binomial expansion negative power pdf
southern regional education board doctoral scholars program
12 major scales eb alto saxophone
chocolate cell phone commercial
john berger about looking pdf
wood carving pdf free
banking exam syllabus pdf
operadores mecanicos pdf
amendments of indian constitution till 2018 pdf
ghost rider emoji
76996697400.pdf
mititas.pdf
rawutoxukowuboli.pdf