Item 226
git_comments:

git_commits:

1. **summary:** KAFKA-6805: Enable broker configs to be stored in ZK before broker start (#4898)
   message: KAFKA-6805: Enable broker configs to be stored in ZK before broker start (#4898) Support configuration of dynamic broker configs in ZooKeeper before starting brokers using ConfigCommand. This will allow password configs to be encrypted and stored in ZooKeeper, without requiring clear passwords in server.properties to bootstrap the broker first. Reviewers: Jason Gustafson <jason@confluent.io>

github_issues:

github_issues_comments:

github_pulls:

1. **title:** KAFKA-6805: Enable broker configs to be stored in ZK before broker start
   body: Support configuration of dynamic broker configs in ZooKeeper before starting brokers using ConfigCommand. This will allow password configs to be encrypted and stored in ZooKeeper, without requiring clear passwords in server.properties to bootstrap the broker first. ### Committer Checklist (excluded from commit message) - [ ] Verify design and implementation - [ ] Verify test coverage and CI build status - [ ] Verify documentation (including upgrade notes)
2. **title:** KAFKA-6805: Enable broker configs to be stored in ZK before broker start
   body: Support configuration of dynamic broker configs in ZooKeeper before starting brokers using ConfigCommand. This will allow password configs to be encrypted and stored in ZooKeeper, without requiring clear passwords in server.properties to bootstrap the broker first. ### Committer Checklist (excluded from commit message) - [ ] Verify design and implementation - [ ] Verify test coverage and CI build status - [ ] Verify documentation (including upgrade notes)
3. **title:** KAFKA-6805: Enable broker configs to be stored in ZK before broker start
   body: Support configuration of dynamic broker configs in ZooKeeper before starting brokers using ConfigCommand. This will allow password configs to be encrypted and stored in ZooKeeper, without requiring clear passwords in server.properties to bootstrap the broker first. ### Committer Checklist (excluded from commit message) - [ ] Verify design and implementation - [ ] Verify test coverage and CI build status - [ ] Verify documentation (including upgrade notes)
4. **title:** KAFKA-6805: Enable broker configs to be stored in ZK before broker start
   body: Support configuration of dynamic broker configs in ZooKeeper before starting brokers using ConfigCommand. This will allow password configs to be encrypted and stored in ZooKeeper, without requiring clear passwords in server.properties to bootstrap the broker first. ### Committer Checklist (excluded from commit message) - [ ] Verify design and implementation - [ ] Verify test coverage and CI build status - [ ] Verify documentation (including upgrade notes)
5. **title:** KAFKA-6805: Enable broker configs to be stored in ZK before broker start
   body: Support configuration of dynamic broker configs in ZooKeeper before starting brokers using ConfigCommand. This will allow password configs to be encrypted and stored in ZooKeeper, without requiring clear passwords in server.properties to bootstrap the broker first. ### Committer Checklist (excluded from commit message) - [ ] Verify design and implementation - [ ] Verify test coverage and CI build status - [ ] Verify documentation (including upgrade notes)
6. **title:** KAFKA-6805: Enable broker configs to be stored in ZK before broker start
   body: Support configuration of dynamic broker configs in ZooKeeper before starting brokers using ConfigCommand. This will allow password configs to be encrypted and stored in ZooKeeper, without requiring clear passwords in server.properties to bootstrap the broker first. ### Committer Checklist (excluded from commit message) - [ ] Verify design and implementation - [ ] Verify test coverage and CI build status - [ ] Verify documentation (including upgrade notes)
7. **title:** KAFKA-6805: Enable broker configs to be stored in ZK before broker start
   body: Support configuration of dynamic broker configs in ZooKeeper before starting brokers using ConfigCommand. This will allow password configs to be encrypted and stored in ZooKeeper, without requiring clear passwords in server.properties to bootstrap the broker first. ### Committer Checklist (excluded from commit message) - [ ] Verify design and implementation - [ ] Verify test coverage and CI build status - [ ] Verify documentation (including upgrade notes)
8. **title:** KAFKA-6805: Enable broker configs to be stored in ZK before broker start
   body: Support configuration of dynamic broker configs in ZooKeeper before starting brokers using ConfigCommand. This will allow password configs to be encrypted and stored in ZooKeeper, without requiring clear passwords in server.properties to bootstrap the broker first. ### Committer Checklist (excluded from commit message) - [ ] Verify design and implementation - [ ] Verify test coverage and CI build status - [ ] Verify documentation (including upgrade notes)
9. **title:** KAFKA-6805: Enable broker configs to be stored in ZK before broker start
   body: Support configuration of dynamic broker configs in ZooKeeper before starting brokers using ConfigCommand. This will allow password configs to be encrypted and stored in ZooKeeper, without requiring clear passwords in server.properties to bootstrap the broker first. ### Committer Checklist (excluded from commit message) - [ ] Verify design and implementation - [ ] Verify test coverage and CI build status - [ ] Verify documentation (including upgrade notes)
10. **title:** KAFKA-6805: Enable broker configs to be stored in ZK before broker start
    body: Support configuration of dynamic broker configs in ZooKeeper before starting brokers using ConfigCommand. This will allow password configs to be encrypted and stored in ZooKeeper, without requiring clear passwords in server.properties to bootstrap the broker first. ### Committer Checklist (excluded from commit message) - [ ] Verify design and implementation - [ ] Verify test coverage and CI build status - [ ] Verify documentation (including upgrade notes)
11. **title:** KAFKA-6805: Enable broker configs to be stored in ZK before broker start
    body: Support configuration of dynamic broker configs in ZooKeeper before starting brokers using ConfigCommand. This will allow password configs to be encrypted and stored in ZooKeeper, without requiring clear passwords in server.properties to bootstrap the broker first. ### Committer Checklist (excluded from commit message) - [ ] Verify design and implementation - [ ] Verify test coverage and CI build status - [ ] Verify documentation (including upgrade notes)

label: code-design
12. **title:** KAFKA-6805: Enable broker configs to be stored in ZK before broker start
body: Support configuration of dynamic broker configs in ZooKeeper before starting brokers using ConfigCommand. This will allow password configs to be encrypted and stored in ZooKeeper, without requiring clear passwords in server.properties to bootstrap the broker first. ### Committer Checklist (excluded from commit message) - [ ] Verify design and implementation - [ ] Verify test coverage and CI build status - [ ] Verify documentation (including upgrade notes)
13. **title:** KAFKA-6805: Enable broker configs to be stored in ZK before broker start
body: Support configuration of dynamic broker configs in ZooKeeper before starting brokers using ConfigCommand. This will allow password configs to be encrypted and stored in ZooKeeper, without requiring clear passwords in server.properties to bootstrap the broker first. ### Committer Checklist (excluded from commit message) - [ ] Verify design and implementation - [ ] Verify test coverage and CI build status - [ ] Verify documentation (including upgrade notes)
14. **title:** KAFKA-6805: Enable broker configs to be stored in ZK before broker start
body: Support configuration of dynamic broker configs in ZooKeeper before starting brokers using ConfigCommand. This will allow password configs to be encrypted and stored in ZooKeeper, without requiring clear passwords in server.properties to bootstrap the broker first. ### Committer Checklist (excluded from commit message) - [ ] Verify design and implementation - [ ] Verify test coverage and CI build status - [ ] Verify documentation (including upgrade notes)
15. **title:** KAFKA-6805: Enable broker configs to be stored in ZK before broker start
body: Support configuration of dynamic broker configs in ZooKeeper before starting brokers using ConfigCommand. This will allow password configs to be encrypted and stored in ZooKeeper, without requiring clear passwords in server.properties to bootstrap the broker first. ### Committer Checklist (excluded from commit message) - [ ] Verify design and implementation - [ ] Verify test coverage and CI build status - [ ] Verify documentation (including upgrade notes)
16. **title:** KAFKA-6805: Enable broker configs to be stored in ZK before broker start
body: Support configuration of dynamic broker configs in ZooKeeper before starting brokers using ConfigCommand. This will allow password configs to be encrypted and stored in ZooKeeper, without requiring clear passwords in server.properties to bootstrap the broker first. ### Committer Checklist (excluded from commit message) - [ ] Verify design and implementation - [ ] Verify test coverage and CI build status - [ ] Verify documentation (including upgrade notes)

github_pulls_comments:

1. @hachikuji KIP-248 is going to add a new ConfigCommand tool that talks to AdminClient. The intention was to deprecate the existing ConfigCommand, but that would prevent us from creating SCRAM credentials (and configuring quotas) before starting up brokers - we need credentials for inter-broker and credentials for AdminClient to be created before you can start up a broker and create other credentials using AdminClient. So the plan is to retain the existing tool to enable this functionality. I was thinking that since we need to retain the tool anyway, we should allow configs to be persisted in ZooKeeper before starting brokers as well. It would enable users to run a script that creates some configs and/or credentials and then start Kafka. Once PR #4904 is merged, this will allow Kafka brokers to be started with no clear passwords in server.properties and only encrypted passwords stored in ZK.
2. @rajinisivaram One other thing. Do you think this is worth mentioning in the upgrade notes?
3. @hachikuji Thanks for the review. I have updated the code, rebased and added to upgrade notes.
4. @hachikuji Thanks for the reviews, merging to trunk and 2.0.

github_pulls_reviews:

1. This seems a little annoying from a user perspective. If we have to open the door for changing configs directly through zk, do we get much benefit from restricting its usage?
2. @hachikuji Thanks for the review. For initial configuration of brokers, we don't do much validation. This feels reasonable since initial config from server.properties is only validated when broker starts up (broker fails to start with invalid config). The validation is slightly better with configs persisted in ZK since we can check the types etc. With dynamic configuration of a running broker, we do a lot more validation. For example, for keystore update, we can check that the file exists on the broker and is a valid store, but that can only be verified by the broker itself. For inter-broker keystore update, we can validate keystore against truststore. We could allow updating using ZK at any time and have the broker print out an error and not apply an invalid config. But since we are able to validate using AdminClient and the tool gives immediate feedback for invalid config, I thought it was better to restrict this for bootstrapping use only.
3. Ok, that makes sense. Perhaps we can explain in the message that configuration through zookeeper is only permitted in order to bootstrap a broker? Maybe we should even mention the password use case?
4. I wonder if we can give the user a better error if the entity name cannot be converted to an integer? For example, we have the `parseBroker` function already in the zk client.
5. Do users need to be able to override these configs?
6. **body:** nit: maybe we should refer to the value `brokerId` here and the line below?
label: code-design
7. Updated to use `parseBroker`.
8. I have made the encoder parameters configurable.
9. Is the part about using the default parameters accurate since they are now configurable?
10. @hachikuji Thank you for the review. Have updated the error message - thanks for noticing that!

jira_issues:

1. **summary:** Allow dynamic broker configs to be configured in ZooKeeper before starting broker
description: At the moment, dynamic broker configs like SSL keystore and password can be configured using ConfigCommand only after a broker is started (using the new AdminClient). To start a broker, these configs have to be defined in server.properties. We want to restrict updates using ZooKeeper once broker starts up, but we should allow updates using ZK prior to starting brokers. This is particularly useful for password configs which are stored encrypted in ZK, making it difficult to set manually before starting brokers. ConfigCommand is being updated to talk to AdminClient under KIP-248, but we will need to maintain the tool using ZK to enable credentials to be created in ZK before starting brokers. So the functionality to set broker configs can sit alongside that.

2. **summary:** Allow dynamic broker configs to be configured in ZooKeeper before starting broker
   description: At the moment, dynamic broker configs like SSL keystore and password can be configured using ConfigCommand only after a broker is started (using the new AdminClient). To start a broker, these configs have to be defined in server.properties. We want to restrict updates using ZooKeeper once broker starts up, but we should allow updates using ZK prior to starting brokers. This is particularly useful for password configs which are stored encrypted in ZK, making it difficult to set manually before starting brokers. ConfigCommand is being updated to talk to AdminClient under KIP-248, but we will need to maintain the tool using ZK to enable credentials to be created in ZK before starting brokers. So the functionality to set broker configs can sit alongside that.
3. **summary:** Allow dynamic broker configs to be configured in ZooKeeper before starting broker
   description: At the moment, dynamic broker configs like SSL keystore and password can be configured using ConfigCommand only after a broker is started (using the new AdminClient). To start a broker, these configs have to be defined in server.properties. We want to restrict updates using ZooKeeper once broker starts up, but we should allow updates using ZK prior to starting brokers. This is particularly useful for password configs which are stored encrypted in ZK, making it difficult to set manually before starting brokers. ConfigCommand is being updated to talk to AdminClient under KIP-248, but we will need to maintain the tool using ZK to enable credentials to be created in ZK before starting brokers. So the functionality to set broker configs can sit alongside that.

jira_issues_comments:

1. rajinisivaram opened a new pull request #4898: KAFKA-6805: Enable broker configs to be stored in ZK before broker start URL: https://github.com/apache/kafka/pull/4898 Support configuration of dynamic broker configs in ZooKeeper before starting brokers using ConfigCommand. This will allow password configs to be encrypted and stored in ZooKeeper, without requiring clear passwords in server.properties to bootstrap the broker first. ### Committer Checklist (excluded from commit message) - [ ] Verify design and implementation - [ ] Verify test coverage and CI build status - [ ] Verify documentation (including upgrade notes) ---------- ---------- This is an automated message from the Apache Git Service. To respond to the message, please log on GitHub and use the URL above to go to the specific comment. For queries about this service, please contact Infrastructure at: users@infra.apache.org
2. Moving this out to 2.1.0 since it is not ready yet.
3. rajinisivaram closed pull request #4898: KAFKA-6805: Enable broker configs to be stored in ZK before broker start URL: https://github.com/apache/kafka/pull/4898 This is a PR merged from a forked repository. As GitHub hides the original diff on merge, it is displayed below for the sake of provenance: As this is a foreign pull request (from a fork), the diff is supplied below (as it won't show otherwise due to GitHub magic): diff --git a/core/src/main/scala/kafka/admin/ConfigCommand.scala b/core/src/main/scala/kafka/admin/ConfigCommand.scala index c1b384fd1fa..6ac0a019dbc 100644 --- a/core/src/main/scala/kafka/admin/ConfigCommand.scala +++ b/core/src/main/scala/kafka/admin/ConfigCommand.scala @@ -24,13 +24,14 @@ import joptsimple._ import kafka.common.Config import kafka.common.InvalidConfigException import kafka.log.LogConfig -import kafka.server.{ConfigEntityName, ConfigType, DynamicConfig} -import kafka.utils.{CommandLineUtils, Exit} +import kafka.server.{ConfigEntityName, ConfigType, Defaults, DynamicBrokerConfig, DynamicConfig, KafkaConfig} +import kafka.utils.{CommandLineUtils, Exit, PasswordEncoder} import kafka.utils.Implicits._ import kafka.zk.{AdminZkClient, KafkaZkClient} import org.apache.kafka.clients.CommonClientConfigs import org.apache.kafka.clients.admin.{AlterConfigsOptions, ConfigEntry, DescribeConfigsOptions, AdminClient => JAdminClient, Config => JConfig} import org.apache.kafka.common.config.ConfigResource +import org.apache.kafka.common.config.types.Password import org.apache.kafka.common.security.JaasUtils import org.apache.kafka.common.security.scram.internals.{ScramCredentialUtils, ScramFormatter, ScramMechanism} import org.apache.kafka.common.utils.{Sanitizer, Time, Utils} @@ -56,11 +57,14 @@ import scala.collection.JavaConverters._ object ConfigCommand extends Config { val DefaultScramIterations = 4096 - // Dynamic broker configs can only be updated using the new AdminClient since they may require - // password encryption currently implemented only in the broker. For consistency with older versions, - // quota-related broker configs can still be updated using ZooKeeper. ConfigCommand will be migrated - // fully to the new AdminClient later (KIP-248). - val BrokerConfigsUpdatableUsingZooKeeper = Set(DynamicConfig.Broker.LeaderReplicationThrottledRateProp, + // Dynamic broker configs can only be updated using the new AdminClient once brokers have started + // so that configs may be fully validated. Prior to starting brokers, updates may be performed using + // ZooKeeper for bootstrapping. This allows all password configs to be stored encrypted in ZK, + // avoiding clear passwords in server.properties. For consistency with older versions, quota-related + // broker configs can still be updated using ZooKeeper at any time. ConfigCommand will be migrated + // to the new AdminClient later for these configs (KIP-248). + val BrokerConfigsUpdatableUsingZooKeeperWhileBrokerRunning = Set(+ DynamicConfig.Broker.LeaderReplicationThrottledRateProp, DynamicConfig.Broker.FollowerReplicationThrottledRateProp, DynamicConfig.Broker.ReplicaAlterLogDirsIoMaxBytesPerSecondProp) @@ -114,9 +118,25 @@ object ConfigCommand extends Config { if (entityType == ConfigType.User) preProcessScramCredentials(configsToBeAdded) - if (entityType == ConfigType.Broker) { - require(configsToBeAdded.asScala.keySet.forall(BrokerConfigsUpdatableUsingZooKeeper.contains), - s"--bootstrap-server option must be specified to update broker configs $configsToBeAdded") + else if (entityType == ConfigType.Broker) { + // Replication quota configs may be updated using ZK at any time. Other dynamic broker configs + // may be updated using ZooKeeper only if the corresponding broker is not running. Dynamic broker + // configs at cluster-default level may be configured using ZK only if there are no brokers running. + val dynamicBrokerConfigs = configsToBeAdded.asScala.keySet.filterNot(BrokerConfigsUpdatableUsingZooKeeperWhileBrokerRunning.contains) + if (dynamicBrokerConfigs.nonEmpty) { + val perBrokerConfig = entityName != ConfigEntityName.Default + val errorMessage = s"--bootstrap-server option must be specified to update broker configs $dynamicBrokerConfigs." + val info = "Broker configuraton updates using ZooKeeper are supported for bootstrapping before brokers" + + " are started to enable encrypted password configs to be stored in ZooKeeper." + if (perBrokerConfig) { + adminZkClient.parseBroker(entityName).foreach { brokerId => + require(zkClient.getBroker(brokerId).isEmpty, s"$errorMessage when broker $entityName is running. $info") + } + } else { + require(zkClient.getAllBrokersInCluster.isEmpty, s"$errorMessage for default cluster if any broker is running. $info") + } + preProcessBrokerConfigs(configsToBeAdded, perBrokerConfig) + } } // compile the final set of configs @@ -156,6 +176,49 @@ object ConfigCommand extends Config { } } + private[admin] def createPasswordEncoder(encoderConfigs: Map[String, String]): PasswordEncoder = { + encoderConfigs.get(KafkaConfig.PasswordEncoderSecretProp) + val encoderSecret = encoderConfigs.getOrElse(KafkaConfig.PasswordEncoderSecretProp, + throw new IllegalArgumentException("Password encoder secret not specified")) + new PasswordEncoder(new Password(encoderSecret), + None,

encoderConfigs.get(KafkaConfig.PasswordEncoderCipherAlgorithmProp).getOrElse(Defaults.PasswordEncoderCipherAlgorithm), +
encoderConfigs.get(KafkaConfig.PasswordEncoderKeyLengthProp).map(_.toInt).getOrElse(Defaults.PasswordEncoderKeyLength), + encoderConfigs.get(KafkaConfig.PasswordEncoderIterationsProp).map(_.toInt).getOrElse(Defaults.PasswordEncoderIterations)) + } + + /** + * Pre-process broker configs provided to convert them to persistent format. + * Password configs are encrypted using the secret `KafkaConfig.PasswordEncoderSecretProp`. + * The secret is removed from `configsToBeAdded` and will not be persisted in ZooKeeper. + */ + private def preProcessBrokerConfigs(configsToBeAdded: Properties, perBrokerConfig: Boolean) { + val passwordEncoderConfigs = new Properties + passwordEncoderConfigs ++= configsToBeAdded.asScala.filterKeys(_.startsWith("password.encoder.")) + if (!passwordEncoderConfigs.isEmpty) { + info(s"Password encoder configs ${passwordEncoderConfigs.keySet} will be used for encrypting" + + " passwords, but will not be stored in ZooKeeper.") + passwordEncoderConfigs.asScala.keySet.foreach(configsToBeAdded.remove) + } + + DynamicBrokerConfig.validateConfigs(configsToBeAdded, perBrokerConfig) + val passwordConfigs = configsToBeAdded.asScala.keySet.filter(DynamicBrokerConfig.isPasswordConfig) + if (passwordConfigs.nonEmpty) { + require(passwordEncoderConfigs.containsKey(KafkaConfig.PasswordEncoderSecretProp), + s"${KafkaConfig.PasswordEncoderSecretProp} must be specified to update $passwordConfigs." + + " Other password encoder configs like cipher algorithm and iterations may also be specified" + + " to override the default encoding parameters. Password encoder configs will not be persisted" + + " in ZooKeeper." +) + + val passwordEncoder = createPasswordEncoder(passwordEncoderConfigs.asScala) + passwordConfigs.foreach { configName => + val encodedValue = passwordEncoder.encode(new Password(configsToBeAdded.getProperty(configName))) + configsToBeAdded.setProperty(configName, encodedValue) + } + } + } + private def describeConfig(zkClient: KafkaZkClient, opts: ConfigCommandOptions, adminZkClient: AdminZkClient) { val configEntity = parseEntity(opts) val describeAllUsers = configEntity.root.entityType == ConfigType.User && !configEntity.root.sanitizedName.isDefined && !configEntity.child.isDefined @@ -358,7 +421,12 @@ object ConfigCommand extends Config { parseQuotaEntity(opts) else { // Exactly one entity type and at-most one entity name expected for other entities - val name = if (opts.options.has(opts.entityName)) Some(opts.options.valueOf(opts.entityName)) else None + val name = if (opts.options.has(opts.entityName)) + Some(opts.options.valueOf(opts.entityName)) + else if (entityTypes.head == ConfigType.Broker && opts.options.has(opts.entityDefault)) + Some(ConfigEntityName.Default) + else + None ConfigEntity(Entity(entityTypes.head, name), None) } } diff --git a/core/src/main/scala/kafka/server/DynamicBrokerConfig.scala b/core/src/main/scala/kafka/server/DynamicBrokerConfig.scala index 4225cdb1a40..72772fa6fcb 100755 --- a/core/src/main/scala/kafka/server/DynamicBrokerConfig.scala +++ b/core/src/main/scala/kafka/server/DynamicBrokerConfig.scala @@ -115,6 +115,43 @@ object DynamicBrokerConfig { } + def validateConfigs(props: Properties, perBrokerConfig: Boolean): Unit = { + def checkInvalidProps(invalidPropNames: Set[String], errorMessage: String): Unit = { + if (invalidPropNames.nonEmpty) + throw new ConfigException(s"$errorMessage: $invalidPropNames") + } + checkInvalidProps(nonDynamicConfigs(props), "Cannot update these configs dynamically") + checkInvalidProps(securityConfigsWithoutListenerPrefix(props), + "These security configs can be dynamically updated only per-listener using the listener prefix") + validateConfigTypes(props) + if (!perBrokerConfig) { + checkInvalidProps(perBrokerConfigs(props), + "Cannot update these configs at default cluster level, broker id must be specified") + } + } + + private def perBrokerConfigs(props: Properties): Set[String] = { + val configNames = props.asScala.keySet + configNames.intersect(PerBrokerConfigs) ++ configNames.filter(ListenerConfigRegex.findFirstIn(_).nonEmpty) + } + + private def nonDynamicConfigs(props: Properties): Set[String] = { + props.asScala.keySet.intersect(DynamicConfig.Broker.nonDynamicProps) + } + + private def securityConfigsWithoutListenerPrefix(props: Properties): Set[String] = { + DynamicSecurityConfigs.filter(props.containsKey) + } + + private def validateConfigTypes(props: Properties): Unit = { + val baseProps = new Properties + props.asScala.foreach { + case (ListenerConfigRegex(baseName), v) => baseProps.put(baseName, v) + case (k, v) => baseProps.put(k, v) + } + DynamicConfig.Broker.validate(baseProps) + } + private[server] def addDynamicConfigs(configDef: ConfigDef): Unit = { KafkaConfig.configKeys.filterKeys(AllDynamicConfigs.contains).values.foreach { config => configDef.define(config.name, config.`type`, config.defaultValue, config.validator, @@ -298,57 +335,26 @@ class DynamicBrokerConfig(private val kafkaConfig: KafkaConfig) extends Logging decoded.foreach { value => props.put(configName, passwordEncoder.encode(new Password(value))) } } } - adminZkClient.changeBrokerConfig(Seq(kafkaConfig.brokerId), props) + adminZkClient.changeBrokerConfig(Some(kafkaConfig.brokerId), props) } } props } private[server] def validate(props: Properties, perBrokerConfig: Boolean): Unit = CoreUtils.inReadLock(lock) { - def checkInvalidProps(invalidPropNames: Set[String], errorMessage: String): Unit = { - if (invalidPropNames.nonEmpty) - throw new ConfigException(s"$errorMessage: $invalidPropNames") - } - checkInvalidProps(nonDynamicConfigs(props), "Cannot update these configs dynamically") - checkInvalidProps(securityConfigsWithoutListenerPrefix(props), - "These security configs can be dynamically updated only per-listener using the listener prefix") - validateConfigTypes(props) + validateConfigs(props, perBrokerConfig) val newProps = mutable.Map[String, String]() newProps ++= staticBrokerConfigs if (perBrokerConfig) { overrideProps(newProps, dynamicDefaultConfigs) overrideProps(newProps, props.asScala) } else { - checkInvalidProps(perBrokerConfigs(props), - "Cannot update these configs at default cluster level, broker id must be specified") overrideProps(newProps, props.asScala) overrideProps(newProps, dynamicBrokerConfigs) } processReconfiguration(newProps, validateOnly = true) } - private def perBrokerConfigs(props: Properties): Set[String] = { - val configNames = props.asScala.keySet - configNames.intersect(PerBrokerConfigs) ++ configNames.filter(ListenerConfigRegex.findFirstIn(_).nonEmpty) - } - - private def nonDynamicConfigs(props: Properties): Set[String] = { - props.asScala.keySet.intersect(DynamicConfig.Broker.nonDynamicProps) - } - - private def securityConfigsWithoutListenerPrefix(props: Properties): Set[String] = { - DynamicSecurityConfigs.filter(props.containsKey) - } - - private def validateConfigTypes(props: Properties): Unit = { - val baseProps = new Properties - props.asScala.foreach { - case (ListenerConfigRegex(baseName), v) => baseProps.put(baseName, v) - case (k, v) => baseProps.put(k, v) - } - DynamicConfig.Broker.validate(baseProps) - } - private def removeInvalidConfigs(props: Properties, perBrokerConfig: Boolean): Unit = { try { validateConfigTypes(props) diff --git a/core/src/main/scala/kafka/zk/AdminZkClient.scala b/core/src/main/scala/kafka/zk/AdminZkClient.scala index 2f8da360c9b..8a6b3ee212d 100644 --- a/core/src/main/scala/kafka/zk/AdminZkClient.scala +++ b/core/src/main/scala/kafka/zk/AdminZkClient.scala @@ -265,6 +265,18 @@ class AdminZkClient(zkClient: KafkaZkClient) extends Logging { } } + def parseBroker(broker: String): Option[Int] = { + broker match { + case ConfigEntityName.Default => None + case _ => + try Some(broker.toInt) + catch { + case _:

NumberFormatException => + throw new IllegalArgumentException(s"Error parsing broker $broker. The broker's Entity Name must be a single integer value") + } + } + } + /** * Change the configs for a given entityType and entityName * @param entityType @@ -273,19 +285,11 @@ class AdminZkClient(zkClient: KafkaZkClient) extends Logging { */ def changeConfigs(entityType: String, entityName: String, configs: Properties): Unit = { - def parseBroker(broker: String): Int = { - try broker.toInt - catch { - case _: NumberFormatException => - throw new IllegalArgumentException(s"Error parsing broker $broker. The broker's Entity Name must be a single integer value") - } - } - entityType match { case ConfigType.Topic => changeTopicConfig(entityName, configs) case ConfigType.Client => changeClientIdConfig(entityName, configs) case ConfigType.User => changeUserOrUserClientIdConfig(entityName, configs) - case ConfigType.Broker => changeBrokerConfig(Seq(parseBroker(entityName)), configs) + case ConfigType.Broker => changeBrokerConfig(parseBroker(entityName), configs) case _ => throw new IllegalArgumentException(s"$entityType is not a known entityType. Should be one of ${ConfigType.Topic}, ${ConfigType.Client}, ${ConfigType.Broker}") } } diff --git a/core/src/test/scala/integration/kafka/server/DynamicBrokerReconfigurationTest.scala b/core/src/test/scala/integration/kafka/server/DynamicBrokerReconfigurationTest.scala index 69ca31703ef..52ad2b9fe8e 100644 --- a/core/src/test/scala/integration/kafka/server/DynamicBrokerReconfigurationTest.scala +++ b/core/src/test/scala/integration/kafka/server/DynamicBrokerReconfigurationTest.scala @@ -126,6 +126,7 @@ class DynamicBrokerReconfigurationTest extends ZooKeeperTestHarness with SaslSet props ++= securityProps(sslProperties1, KEYSTORE_PROPS, listenerPrefix(SecureExternal)) val kafkaConfig = KafkaConfig.fromProps(props) + configureDynamicKeystoreInZooKeeper(kafkaConfig, sslProperties1) servers += TestUtils.createServer(kafkaConfig) } @@ -778,21 +779,12 @@ class DynamicBrokerReconfigurationTest extends ZooKeeperTestHarness with SaslSet val props = adminZkClient.fetchEntityConfig(ConfigType.Broker, server.config.brokerId.toString) val propsEncodedWithOldSecret = props.clone().asInstanceOf[Properties] val config = server.config - val secret = config.passwordEncoderSecret.getOrElse(throw new IllegalStateException("Password encoder secret not configured")) val oldSecret = "old-dynamic-config-secret" config.dynamicConfig.staticBrokerConfigs.put(KafkaConfig.PasswordEncoderOldSecretProp, oldSecret) val passwordConfigs = props.asScala.filterKeys(DynamicBrokerConfig.isPasswordConfig) assertTrue("Password configs not found", passwordConfigs.nonEmpty) - val passwordDecoder = new PasswordEncoder(secret, - config.passwordEncoderKeyFactoryAlgorithm, - config.passwordEncoderCipherAlgorithm, - config.passwordEncoderKeyLength, - config.passwordEncoderIterations) - val passwordEncoder = new PasswordEncoder(new Password(oldSecret), - config.passwordEncoderKeyFactoryAlgorithm, - config.passwordEncoderCipherAlgorithm, - config.passwordEncoderKeyLength, - config.passwordEncoderIterations) + val passwordDecoder = createPasswordEncoder(config, config.passwordEncoderSecret) + val passwordEncoder = createPasswordEncoder(config, Some(new Password(oldSecret))) passwordConfigs.foreach { case (name, value) => val decoded = passwordDecoder.decode(value).value propsEncodedWithOldSecret.put(name, passwordEncoder.encode(new Password(decoded))) @@ -1161,12 +1153,39 @@ class DynamicBrokerReconfigurationTest extends ZooKeeperTestHarness with SaslSet private def listenerPrefix(name: String): String = new ListenerName(name).configPrefix - private def configureDynamicKeystoreInZooKeeper(kafkaConfig: KafkaConfig, brokers: Seq[Int], sslProperties: Properties): Unit = { + private def configureDynamicKeystoreInZooKeeper(kafkaConfig: KafkaConfig, sslProperties: Properties): Unit = { + val externalListenerPrefix = listenerPrefix(SecureExternal) val sslStoreProps = new Properties - sslStoreProps ++= securityProps(sslProperties, KEYSTORE_PROPS, listenerPrefix(SecureExternal)) - val persistentProps = kafkaConfig.dynamicConfig.toPersistentProps(sslStoreProps, perBrokerConfig = true) + sslStoreProps ++= securityProps(sslProperties, KEYSTORE_PROPS, externalListenerPrefix) + sslStoreProps.put(KafkaConfig.PasswordEncoderSecretProp, kafkaConfig.passwordEncoderSecret.map(_.value).orNull) zkClient.makeSurePersistentPathExists(ConfigEntityChangeNotificationZNode.path) - adminZkClient.changeBrokerConfig(brokers, persistentProps) + + val args = Array("--zookeeper", kafkaConfig.zkConnect, + "--alter", "--add-config", sslStoreProps.asScala.map { case (k, v) => s"$k=$v" }.mkString(","), + "--entity-type", "brokers", + "--entity-name", kafkaConfig.brokerId.toString) + ConfigCommand.main(args) + + val passwordEncoder = createPasswordEncoder(kafkaConfig, kafkaConfig.passwordEncoderSecret) + val brokerProps = adminZkClient.fetchEntityConfig("brokers", kafkaConfig.brokerId.toString) + assertEquals(4, brokerProps.size) + assertEquals(sslProperties.get(SSL_KEYSTORE_TYPE_CONFIG), + brokerProps.getProperty(s"$externalListenerPrefix$SSL_KEYSTORE_TYPE_CONFIG")) + assertEquals(sslProperties.get(SSL_KEYSTORE_LOCATION_CONFIG), + brokerProps.getProperty(s"$externalListenerPrefix$SSL_KEYSTORE_LOCATION_CONFIG")) + assertEquals(sslProperties.get(SSL_KEYSTORE_PASSWORD_CONFIG), + passwordEncoder.decode(brokerProps.getProperty(s"$externalListenerPrefix$SSL_KEYSTORE_PASSWORD_CONFIG"))) + assertEquals(sslProperties.get(SSL_KEY_PASSWORD_CONFIG), + passwordEncoder.decode(brokerProps.getProperty(s"$externalListenerPrefix$SSL_KEY_PASSWORD_CONFIG"))) } + + private def createPasswordEncoder(config: KafkaConfig, secret: Option[Password]): PasswordEncoder = { + val encoderSecret = secret.getOrElse(throw new IllegalStateException("Password encoder secret not configured")) + new PasswordEncoder(encoderSecret, + config.passwordEncoderKeyFactoryAlgorithm, + config.passwordEncoderCipherAlgorithm, + config.passwordEncoderKeyLength, + config.passwordEncoderIterations) + } private def waitForConfig(propName: String, propValue: String, maxWaitMs: Long = 10000): Unit = { diff --git a/core/src/test/scala/unit/kafka/admin/ConfigCommandTest.scala b/core/src/test/scala/unit/kafka/admin/ConfigCommandTest.scala index a24800f3cea..2644dcce6bf 100644 --- a/core/src/test/scala/unit/kafka/admin/ConfigCommandTest.scala +++ b/core/src/test/scala/unit/kafka/admin/ConfigCommandTest.scala @@ -20,21 +20,25 @@ import java.util import java.util.Properties import kafka.admin.ConfigCommand.ConfigCommandOptions +import kafka.api.ApiVersion import kafka.cluster.{Broker, EndPoint} import kafka.common.InvalidConfigException -import kafka.server.ConfigEntityName +import kafka.server.{ConfigEntityName, KafkaConfig} import kafka.utils.{Exit, Logging} -import kafka.zk.{AdminZkClient, KafkaZkClient, ZooKeeperTestHarness} +import kafka.zk.{AdminZkClient, BrokerInfo, KafkaZkClient, ZooKeeperTestHarness} import org.apache.kafka.clients.admin._ -import org.apache.kafka.common.config.ConfigResource +import org.apache.kafka.common.config.{ConfigException, ConfigResource} import org.apache.kafka.common.internals.KafkaFutureImpl +import org.apache.kafka.common.Node +import org.apache.kafka.common.network.ListenerName +import org.apache.kafka.common.security.auth.SecurityProtocol import org.apache.kafka.common.security.scram.internals.ScramCredentialUtils import org.apache.kafka.common.utils.Sanitizer import org.easymock.EasyMock import org.junit.Assert._ import org.junit.Test -import scala.collection.mutable

{Seq, mutable} import scala.collection.JavaConverters._ class ConfigCommandTest extends ZooKeeperTestHarness with Logging { @@ -51,7 +55,7 @@ class ConfigCommandTest extends ZooKeeperTestHarness with Logging { "--entity-name", "1", "--entity-type", "brokers", "--alter", - "--add-config", "message.max.size=100000")) + "--add-config", "security.inter.broker.protocol=PLAINTEXT")) } @Test @@ -306,14 +310,99 @@ class ConfigCommandTest extends ZooKeeperTestHarness with Logging { ConfigCommand.alterConfig(null, createOpts, new DummyAdminZkClient(zkClient)) } - @Test (expected = classOf[IllegalArgumentException]) - def shouldNotUpdateDynamicBrokerConfigUsingZooKeeper(): Unit = { - val createOpts = new ConfigCommandOptions(Array("--zookeeper", zkConnect, - "--entity-name", "1", - "--entity-type", "brokers", - "--alter", - "--add-config", "message.max.size=100000")) - ConfigCommand.alterConfig(null, createOpts, new DummyAdminZkClient(zkClient)) + @Test + def testDynamicBrokerConfigUpdateUsingZooKeeper(): Unit = { + val brokerId = "1" + val adminZkClient = new AdminZkClient(zkClient) + val alterOpts = Array("--zookeeper", zkConnect, "--entity-type", "brokers", "--alter") + + def entityOpt(brokerId: Option[String]): Array[String] = { + brokerId.map(id => Array("--entity-name", id)).getOrElse(Array("--entity-default")) + } + + def alterConfig(configs: Map[String, String], brokerId: Option[String], + encoderConfigs: Map[String, String] = Map.empty): Unit = { + val configStr = (configs ++ encoderConfigs).map { case (k, v) => s"$k=$v" }.mkString(",") + val addOpts = new ConfigCommandOptions(alterOpts ++ entityOpt(brokerId) ++ Array("--add-config", configStr)) + ConfigCommand.alterConfig(zkClient, addOpts, adminZkClient) + } + + def verifyConfig(configs: Map[String, String], brokerId: Option[String]): Unit = { + val entityConfigs = zkClient.getEntityConfigs("brokers", brokerId.getOrElse(ConfigEntityName.Default)) + assertEquals(configs, entityConfigs.asScala) + } + + def alterAndVerifyConfig(configs: Map[String, String], brokerId: Option[String]): Unit = { + alterConfig(configs, brokerId) + verifyConfig(configs, brokerId) + } + + def deleteAndVerifyConfig(configNames: Set[String], brokerId: Option[String]): Unit = { + val deleteOpts = new ConfigCommandOptions(alterOpts ++ entityOpt(brokerId) ++ + Array("--delete-config", configNames.mkString(","))) + ConfigCommand.alterConfig(zkClient, deleteOpts, adminZkClient) + verifyConfig(Map.empty, brokerId) + } + + // Add config + alterAndVerifyConfig(Map("message.max.size" -> "110000"), Some(brokerId)) + alterAndVerifyConfig(Map("message.max.size" -> "120000"), None) + + // Change config + alterAndVerifyConfig(Map("message.max.size" -> "130000"), Some(brokerId)) + alterAndVerifyConfig(Map("message.max.size" -> "140000"), None) + + // Delete config + deleteAndVerifyConfig(Set("message.max.size"), Some(brokerId)) + deleteAndVerifyConfig(Set("message.max.size"), None) + + // Listener configs: should work only with listener name + alterAndVerifyConfig(Map("listener.name.external.ssl.keystore.location" -> "/tmp/test.jks"), Some(brokerId)) + intercept[ConfigException](alterConfig(Map("ssl.keystore.location" -> "/tmp/test.jks"), Some(brokerId))) + + // Per-broker config configured at default cluster-level should fail + intercept[ConfigException](alterConfig(Map("listener.name.external.ssl.keystore.location" -> "/tmp/test.jks"), None)) + deleteAndVerifyConfig(Set("listener.name.external.ssl.keystore.location"), Some(brokerId)) + + // Password config update without encoder secret should fail + intercept[IllegalArgumentException](alterConfig(Map("listener.name.external.ssl.keystore.password" -> "secret"), Some(brokerId))) + + // Password config update with encoder secret should succeed and encoded password must be stored in ZK + val configs = Map("listener.name.external.ssl.keystore.password" -> "secret", "log.cleaner.threads" -> "2") + val encoderConfigs = Map(KafkaConfig.PasswordEncoderSecretProp -> "encoder-secret") + alterConfig(configs, Some(brokerId), encoderConfigs) + val brokerConfigs = zkClient.getEntityConfigs("brokers", brokerId) + assertFalse("Encoder secret stored in ZooKeeper", brokerConfigs.contains(KafkaConfig.PasswordEncoderSecretProp)) + assertEquals("2", brokerConfigs.getProperty("log.cleaner.threads")) // not encoded + val encodedPassword = brokerConfigs.getProperty("listener.name.external.ssl.keystore.password") + val passwordEncoder = ConfigCommand.createPasswordEncoder(encoderConfigs) + assertEquals("secret", passwordEncoder.decode(encodedPassword).value) + assertEquals(configs.size, brokerConfigs.size) + + // Password config update with overrides for encoder parameters + val configs2 = Map("listener.name.internal.ssl.keystore.password" -> "secret2") + val encoderConfigs2 = Map(KafkaConfig.PasswordEncoderSecretProp -> "encoder-secret", + KafkaConfig.PasswordEncoderCipherAlgorithmProp -> "DES/CBC/PKCS5Padding", + KafkaConfig.PasswordEncoderIterationsProp -> "1024", + KafkaConfig.PasswordEncoderKeyFactoryAlgorithmProp -> "PBKDF2WithHmacSHA1", + KafkaConfig.PasswordEncoderKeyLengthProp -> "64") + alterConfig(configs2, Some(brokerId), encoderConfigs2) + val brokerConfigs2 = zkClient.getEntityConfigs("brokers", brokerId) + val encodedPassword2 = brokerConfigs2.getProperty("listener.name.internal.ssl.keystore.password") + assertEquals("secret2", ConfigCommand.createPasswordEncoder(encoderConfigs).decode(encodedPassword2).value) + assertEquals("secret2", ConfigCommand.createPasswordEncoder(encoderConfigs2).decode(encodedPassword2).value) + + // Password config update at default cluster-level should fail + intercept[ConfigException](alterConfig(configs, None, encoderConfigs)) + + // Dynamic config updates using ZK should fail if broker is running. + registerBrokerInZk(brokerId.toInt) + intercept[IllegalArgumentException](alterConfig(Map("message.max.size" -> "210000"), Some(brokerId))) + intercept[IllegalArgumentException](alterConfig(Map("message.max.size" -> "220000"), None)) + + // Dynamic config updates using ZK should for a different broker that is not running should succeed + alterAndVerifyConfig(Map("message.max.size" -> "230000"), Some("2")) } @Test (expected = classOf[IllegalArgumentException]) @@ -322,7 +411,7 @@ class ConfigCommandTest extends ZooKeeperTestHarness with Logging { "--entity-name", "1", "--entity-type", "brokers", "--alter", - "--add-config", "a=")) + "--add-config", "a==")) ConfigCommand.alterConfig(null, createOpts, new DummyAdminZkClient(zkClient)) } @@ -593,6 +682,14 @@ class ConfigCommandTest extends ZooKeeperTestHarness with Logging { Seq("<default>/clients/client-3", sanitizedPrincipal + "/clients/client-2")) } + private def registerBrokerInZk(id: Int): Unit = { + zkClient.createTopLevelPaths() + val securityProtocol = SecurityProtocol.PLAINTEXT + val endpoint = new EndPoint("localhost", 9092, ListenerName.forSecurityProtocol(securityProtocol), securityProtocol) + val brokerInfo = BrokerInfo(Broker(id, Seq(endpoint), rack = None), ApiVersion.latestVersion, jmxPort = 9192) + zkClient.registerBrokerInZk(brokerInfo) + } + class DummyAdminZkClient(zkClient: KafkaZkClient) extends AdminZkClient(zkClient) { override def changeBrokerConfig(brokerIds: Seq[Int], configs: Properties): Unit = {} override def fetchEntityConfig(entityType: String, entityName: String): Properties = {new Properties} diff --git a/docs/configuration.html b/docs/configuration.html index 8c86534fb15..90c990bdaad 100644 --- a/docs/configuration.html +++ b/docs/configuration.html @@ -90,6 +90,23 @@ <h5>Updating Password Configs Dynamically</h5> using <code>kafka-configs.sh</code> even if the password config is not being altered. This constraint will be removed in a future release.</p> + <h5>Updating Password Configs in ZooKeeper Before Starting Brokers</h5> + + From Kafka 2.0.0 onwards, <code>kafka-configs.sh</code> enables dynamic broker configs to be updated using ZooKeeper before + starting brokers for bootstrapping. This enables all password configs to be stored in encrypted form, avoiding the need for + clear passwords in <code>server.properties</code>. The broker config

<code>password.encoder.secret</code> must also be specified + if any password configs are included in the alter command. Additional encryption parameters may also be specified. Password + encoder configs will not be persisted in ZooKeeper. For example, to store SSL key password for listener <code>INTERNAL</code> + on broker 0: + + <pre class="brush: bash;"> + > bin/kafka-configs.sh --zookeeper localhost:2181 --entity-type brokers --entity-name 0 --alter --add-config + 'listener.name.internal.ssl.key.password=key-password,password.encoder.secret=secret,password.encoder.iterations=8192' + </pre> + + The configuration <code>listener.name.internal.ssl.key.password</code> will be persisted in ZooKeeper in encrypted + form using the provided encoder configs. The encoder secret and iterations are not persisted in ZooKeeper. + <h5>Updating SSL Keystore of an Existing Listener</h5> Brokers may be configured with SSL keystores with short validity periods to reduce the risk of compromised certificates. Keystores may be updated dynamically without restarting the broker. The config name must be prefixed with the listener prefix diff --git a/docs/upgrade.html b/docs/upgrade.html index 89c90d19d26..13498364b62 100644 --- a/docs/upgrade.html +++ b/docs/upgrade.html @@ -127,6 +127,8 @@ <h5><a id="upgrade_200_notable" href="#upgrade_200_notable">Notable changes in 2 KIP-283 also adds new topic and broker configurations <code>message.downconversion.enable</code> and <code>log.message.downconversion.enable</code> respectively to control whether down-conversion is enabled. When disabled, broker does not perform any down-conversion and instead sends an <code>UNSUPPORTED_VERSION</code> error to the client. + Dynamic broker configuration options can be stored in ZooKeeper using kafka-configs.sh before brokers are started. + This option can be used to avoid storing clear passwords in server.properties as all password configs may be stored encrypted in ZooKeeper. <h5>New Protocol Versions</h5> ---------- ---------- This is an automated message from the Apache Git Service. To respond to the message, please log on GitHub and use the URL above to go to the specific comment. For queries about this service, please contact Infrastructure at: users@infra.apache.org