# Руководство администратора
# «РЕД V»

## СОДЕРЖАНИЕ

# ВВЕДЕНИЕ

Руководство предназначено для использования администраторами Synapse («глобальными администраторами»). Администраторами Synapse обычно являются опытные пользователи Synapse с правами администратора в Cortex, которые отвечают за настройку и управление промышленным экземпляром Synapse.

Руководство администратора Synapse содержит важные инструкции и справочную информацию по темам, связанным с повседневными административными задачами Synapse, и сосредоточено на использовании Storm для выполнения этих задач.

Synapse предоставляет ряд дополнительных методов, которые можно использовать для выполнения некоторых или всех задач, описанных в этом руководстве; однако эти методы здесь не рассматриваются. Дополнительные методы включают:

- Библиотеки Storm, которые позволяют работать с широким спектром объектов в Synapse.
- Инструменты Synapse, которые можно использовать из командной строки хоста (в отличие от командной строки Storm). Инструменты доступны в synapse.tools Synapse Python API. Руководство пользователя Synapse содержит документацию по некоторым из этих инструментов.
- Synapse HTTP/REST API.

**Совет:** Если Вы являетесь коммерческим пользователем Synapse UI (Optic), ознакомьтесь с документацией по пользовательскому интерфейсу для получения информации о выполнении задач администрирования Synapse с помощью Optic. Optic упрощает многие административные задачи Synapse. Однако рекомендуется ознакомиться с информацией, содержащейся в этом руководстве, для получения важной справочной информации и общего обзора соответствующих тем.

# 1 ВКЛЮЧЕНИЕ ДОПОЛНИТЕЛЬНЫХ ФУНКЦИЙ SYNAPSE

Проект Vertex предоставляет ряд дополнительных функций, расширяющих функциональность Synapse. Дополнительную информацию о настройке Cortex для использования быстрых дополнительных функций см. в блоге, посвященном дополнительным функциям Synapse.

**Примечание:** Расширенные функции развертываются через собственные контейнеры Docker и обычно настраиваются командой DevOps.

# 2 СОЗДАНИЕ И УПРАВЛЕНИЕ ПОЛЬЗОВАТЕЛЯМИ И РОЛЯМИ

Для аутентификации и доступа к Synapse требуется учетная запись пользователя. Наличие «учетной записи Synapse» фактически означает наличие учетной записи в Cortex.

В Synapse роль может использоваться для «группировки» пользователей со схожими обязанностями (и соответствующими требованиями к разрешениям). Пользователю можно предоставить или отозвать одну или несколько ролей.

Предоставление (или отказ) в разрешениях пользователям или ролям осуществляется путем назначения правил, которые определяют эти разрешения (см. п. Назначение разрешений и управление ими).

В состав Synapse входят следующие встроенные пользователи и роли:

- Пользователь Root. Учетная запись root имеет права администратора в Cortex. Статус администратора учетной записи root нельзя отозвать, а учетную запись нельзя заблокировать/отключить.
- Роль all. Роль all имеет доступ на чтение к Cortex (в частности, к любому представлению с параметром worldreadable = true, который включает представление по умолчанию). Всем учетным записям пользователей автоматически присваивается роль all (они являются частью «группы» all); эта роль не может быть отозвана.

**Совет:** Набор команд Storm auth используется совместно для управления пользователями, ролями и разрешениями в Storm.

В коммерческом пользовательском интерфейсе Optic пользователями, ролями и разрешениями можно управлять с помощью инструмента администрирования и диалоговых окон, связанных с различными объектами (например, представлениями или историями).

**Примечание:** В приведенных ниже описаниях и примерах предполагается, что Synapse развернут с использованием встроенного управления Synapse и аутентификации пользователей, ролей и разрешений. Руководство Synapse Devops содержит информацию о подготовке первоначальных пользователей при первом развертывании Synapse (см. раздел Назначение разрешений и управление ими). Это руководство сфокусировано на постоянном управлении пользователями и ролями после того, как администраторы Synapse получат доступ к Storm (например, к CLI Storm или Optic UI).

## 2.1 РАБОТА С ПОЛЬЗОВАТЕЛЯМИ

### 2.1.1 ДОБАВЛЕНИЕ ПОЛЬЗОВАТЕЛЯ

Команда auth.user.add создает нового пользователя. У недавно созданных пользователей нет никаких разрешений (кроме тех, которые связаны со встроенной ролью all).

Например:

Добавим пользователя «Ron» с адресом электронной почты ronthecat@vertex.link:

```
storm> auth.user.add ron --email ronthecat@vertex.link
User (ron) added with iden: c623a3e121c0442cc2673c6d6458ad6a
```

**Совет:** Пользователи представлены уникальным 128-битным идентификатором (iden). Можно изменять информацию об учетной записи пользователя (например, имя пользователя или связанный с ним адрес электронной почты), не затрагивая базовый идентификатор или какие-либо связанные с ним роли или разрешения.

### 2.1.2 Отображение пользователя

Команда auth.user.show отображает информацию о пользователе, включая любые назначенные роли или правила (разрешения) и их порядок.

Например, выведем информацию для пользователя «Ron»:

```
storm> auth.user.show ron
User: ron (c623a3e121c0442cc2673c6d6458ad6a)

  Locked: false
  Admin: false
  Email: ronthecat@vertex.link
  Rules:

  Roles:
    d272a0db0ffa059d179484777a9405a3 - all

  Gates:
```

### 2.1.3 Изменение пользователя

Команда auth.user.mod изменяет учетную запись пользователя. Воспользуйтесь этой командой, чтобы:

- изменить имя пользователя или адрес электронной почты, связанные с пользователем;
- установить или сбросить пароль пользователя;
- присвоить пользователю статус администратора (или удалить его);
- заблокировать (или разблокировать) учетную запись.

Например:

1. Обновим адрес электронной почты для пользователя «Ron»:

```
storm> auth.user.mod ron --email ron@vertex.link
User (ron) email address set to ron@vertex.link.
```

2. Присвоим статус администратора пользователю «ron_admin»:

```
storm> auth.user.mod ron_admin --admin $lib.true
User (ron_admin) admin status set to true.
```

3. Удалим статус администратора у пользователя «ron_admin»:

```
storm> auth.user.mod ron_admin --admin $lib.false
User (ron_admin) admin status set to false.
```

4. Заблокируем аккаунт пользователя «ron_admin»:

```
storm> auth.user.mod ron_admin --locked $lib.true
User (ron_admin) locked status set to true.
```

**Внимание!** Рекомендуется при необходимости блокировать (отключать) учетные записи, а не удалять их. Изменения данных в Cortex (например, создание узлов, настройка свойств или добавление тегов) связаны с учетной записью пользователя, которая внесла эти изменения. Удаление учетной записи, связанной с прошлыми изменениями, не позволит идентифицировать пользователя, который внес эти изменения.

При необходимости учетные записи пользователей можно удалить с помощью библиотеки $lib.auth.users.del(iden), но эквивалентной команды Storm нет.

### 2.1.4 Вывод списка всех пользователей

Команда auth.user.list выводит список всех пользователей в Cortex.

Например, выведем список всех пользователей:

```
storm> auth.user.list
Users:
  ron
  root

Locked Users:
  ron_admin
```

## 2.2 Работа с ролями

### 2.2.1 Добавление роли

Команда auth.role.add создает новую роль. У вновь созданных ролей нет никаких разрешений или связанных учетных записей пользователей.

Например, добавим новую роль «cattribution analyst»:

```
storm> auth.role.add "cattribution analyst"
Role (cattribution analyst) added with iden:
d2251abb9462dc2e0374ade46e23834d
```

**Совет:** Роли представлены уникальным 128-битным идентификатором (iden). Можно изменять информацию о роли (например, имя роли), не затрагивая базовую роль или какие-либо связанные с ней разрешения или пользователей.

### 2.2.2 Отображение роли

Команда auth.role.show отображает информацию о роли, включая все назначенные правила (разрешения) и связанные с ними объекты.

Например, выведем информацию о роли «all»:

```
storm> auth.role.show all
Role: all (d272a0db0ffa059d179484777a9405a3)

  Rules:

  Gates:
    ec0d9ed9e3a8e868471589de7795d5be - (layer)
      [0  ] - layer.read
    e478f1b28af0d45ccbdfdc5346b369d1 - (view)
      [0  ] - view.read
```

### 2.2.3 Изменение роли

Команда auth.role.mod изменяет роль. С помощью этой команды можно изменить название роли.

Например, изменим название роли «cattribution analyst» на «meow-ware analyst»:

```
storm> auth.role.mod "cattribution analyst" --name "meow-ware analyst"
Role (cattribution analyst) renamed to meow-ware analyst.
```

### 2.2.4 Вывод списка ролей

Команда auth.role.list выводит список всех ролей в Cortex.

Например, выведем список всех ролей:

```
storm> auth.role.list
Roles:
  a-cat-emic researcher
  all
  cattribution analyst
  meow-ware analyst
```

### 2.2.5 Удаление роли

Команда auth.role.del удаляет роль.

Например, удалим роль «meow-ware analyst»:

```
storm> auth.role.del "meow-ware analyst"
Role (meow-ware analyst) deleted.
```

**Примечание:** Удаление роли никак не влияет на пользователей, которым была предоставлена эта роль (за исключением потери всех разрешений, предоставляемых этой ролью). Учетные записи пользователей остаются неизменными, а роль просто удаляется из списка ролей каждого пользователя.

## 2.3 Предоставление или отзыв ролей

Предоставление роли пользователю позволяет ему наследовать права доступа роли. При отзыве роли у пользователя удаляются соответствующие права доступа. Невозможно одну роль предоставить другой роли (т. е. роли не могут быть вложенными).

Предоставлять или отзывать роли можно с помощью команд auth.user.grant и auth.user.revoke.

Например:

1. Предоставим роль «cattribution analyst» пользователю «ron»:

```
storm> auth.user.grant ron "cattribution analyst"
Granting role cattribution analyst to user ron.
```

2. Отзовем роль «a-cat-emic researcher» у пользователя «ron»:

```
storm> auth.user.revoke ron "a-cat-emic researcher"
Revoking role a-cat-emic researcher from user ron.
```

**Примечание:** Порядок, в котором роли предоставляются пользователю, имеет значение; при определении того, есть ли у пользователя разрешение на выполнение того или иного действия, последовательно проверяются разрешения для каждой из ролей пользователя.

Каждая роль, предоставленная пользователю, добавляется в конец набора ролей, если не указано местоположение (индекс) для роли. Чтобы «упорядочить» роли, необходимо либо:

- отозвать роли и предоставить их в нужном порядке;
- использовать параметр --index, чтобы указать местоположение для вставки роли;
- использовать setRoles(idens), чтобы заменить роли пользователя новым списком ролей;

- использовать коммерческий пользовательский интерфейс Synapse (Optic), чтобы изменить порядок ролей с помощью перетаскивания.

Дополнительные сведения о разрешениях и приоритете см. в пункте Общие сведения о разрешениях.

# 3 НАЗНАЧЕНИЕ РАЗРЕШЕНИЙ И УПРАВЛЕНИЕ ИМИ

Synapse предоставляет очень гибкую систему управления доступом на основе ролей (RBAC). Правила используются для назначения разрешений пользователям и/или ролям с определенным порядком приоритета для оценки разрешений.

Разрешения могут быть назначены очень широко, например, разрешать пользователю (или роли) создавать / изменять / удалять любой узел. Разрешения также могут быть очень детализированными, ограничивая пользователей так, что они могут создавать только определенные узлы, устанавливать определенные свойства, создавать определенные ребра или применять определенные теги.

## 3.1 Общие сведения о разрешениях

Прежде чем описывать, как назначать разрешения в Synapse и управлять ими, полезно определить некоторые ключевые компоненты Synapse и экосистемы разрешений.

### 3.1.1 Службы

Synapse разработан как модульный набор служб. Службу можно рассматривать как контейнер, используемый для запуска приложения. Службы Synapse составляют ядро архитектуры Synapse и включают в себя Cortex (хранилище данных), Axon (хранилище файлов) и коммерческий интерфейс Optic UI. Службы обрабатывают аутентификацию и авторизацию пользователей.

С точки зрения администратора Synapse, Вы в первую очередь будете заниматься управлением учетными записями пользователей и разрешениями на доступ к Synapse Cortex (и внутри него).

**Совет:** Когда мы говорим о «пользователях Synapse» или «разрешениях для Synapse», мы обычно имеем в виду учетные записи пользователей и роли в Cortex, а также разрешения для Cortex и связанных с ним объектов.

В зависимости от развертывания Synapse может потребоваться предоставить разрешения для дополнительных служб Synapse или управлять ими. Более подробную информацию см. в пунктах Разрешения Optic и Разрешения на Power-Up.

### 3.1.2 Cortex

Cortex — это основное хранилище данных Synapse. Пользователи и роли создаются и управляются в Cortex, и большинство действий, для которых пользователям потребуются разрешения, применяются к Cortex и к представлениям, слоям и данным (узлам, тегам и т. д.), которые там находятся.

### 3.1.3 Шлюз аутентификации

Шлюз аутентификации (или «gate», неформально) — это объект в службе (например, Cortex), который может иметь свой собственный набор разрешений. View и Layer являются распространенными примерами шлюзов аутентификации.

Шлюзы аутентификации представлены 128-битным идентификатором (ident), который уникально идентифицирует сам объект управления аутентификацией. У них также есть связанный тип для указания вида объекта управления аутентификацией (например, «view»). Некоторые шлюзы аутентификации также поддерживают использование «удобных для пользователя» имен, хотя это зависит от типа шлюза аутентификации и не влияет на базовый идентификатор или связанные разрешения.

### 3.1.4 Область действия

Область действия относится к объекту, к которому применяется определенное разрешение. Например, разрешения, предоставленные шлюзу аутентификации (например, представление), ограничены (или локальны) для этого шлюза аутентификации. Разрешения, предоставляемые на уровне Cortex, являются глобальными по отношению к Cortex.

Область действия влияет на порядок (приоритет), в котором оцениваются разрешения.

### 3.1.5 Разрешение

Разрешение — это строка, которая используется для управления доступом. Например:

```
view.add
```

**Совет:** Список большинства разрешений, доступных в Cortex, можно найти в разделе Разрешения Cortex. Также можно отобразить список в Synapse, используя команду auth.list.perms.

В большинстве строк разрешений используется точечный (иерархический) формат; указание разрешения, расположенного выше по иерархии, включает в себя все разрешения, расположенные ниже него. Например, разрешение view включает в себя все следующие разрешения: view.add, view.del, view.read и view.set.

Разрешения, относящиеся к таким объектам, как узлы или теги, могут при необходимости расширять строку разрешений, делая ее более конкретной, ссылаясь на определенные формы, свойства, теги/деревья тегов или границы раздела. Это позволяет устанавливать разрешения с высокой степенью детализации.

Детализированные разрешения могут быть полезны для организаций со специализированными пользователями или командами, где определенные лица отвечают за определенные виды анализа (например, стратегический анализ или отслеживание тактических угроз) и должны быть единственными пользователями, имеющими право создавать, изменять и помечать тегами определенные типы данных.

Детализированные разрешения также могут использоваться для разграничения ролей старшего и младшего персонала; например, только старшим аналитикам может быть разрешено применять теги, представляющие определенные оценки (например, определение).

Примеры детализированных разрешений представлены в таблице 1.

Таблица 1 - Виды разрешений

| **Описание** | **Разрешение** |
|---|---|
| Выполнение любых действий с любым типом узла (включая удаление узлов и работу со свойствами, тегами, ребрами или данными узлов) | node |
| Добавление любого типа узла (но не удаление узла или не работа со свойствами, тегами, ребрами или данными узла ) | node.add |
| Только добавление узлов inet:ipv4 (но не установка свойства или не работа с тегами или ребрами) | node.add.inet:ipv4 |
| Только добавление (установка) свойства :asn для узлов inet:ipv4 (но не создание узлов или работа с другими свойствами, тегами, ребрами и т.д.) | node.prop.set.inet:ipv4:asn |
| Добавление или удаление любого тега (Обратите внимание, что добавление/удаление тегов может потребовать возможность создания узлов syn:tag, если только эти узлы еще не существуют) | node.tag |
| Добавление и удаление тегов только в дереве тегов mytag | node.tag.add.mytag<br>node.tag.del.mytag |
| Добавление или удаление любого ребра (Обратите внимание, что добавление или удаление ребер позволяет создавать ребра между любыми узлами; ограничений модели на типы узлов, которые могут быть соединены, нет. Это также позволяет создавать новые/произвольно названные ребра.) | node.edge |
| Только добавление ребра | node.edge.add |
| Добавление только ссылок на ребра | node.edge.add.refs |

**Примечание:** Строки разрешений не поддерживают символы подстановки (*). Например, нельзя указать node.tag.*.mytag, чтобы разрешить пользователям добавлять и удалять теги в дереве mytag.

### 3.1.6 Правило

Правило используется для предоставления (или запрета) определенного разрешения. Правила оцениваются в определенном порядке приоритета.

При указании правила используется неявная директива allow; строка разрешения сама по себе указывает, что разрешение разрешено/истинно:

```
view.add
```

Чтобы использовать правило для запрета разрешения, необходимо использовать символ «не» или «восклицательный знак» (!), чтобы указать, что разрешение отклонено/неверно:

```
!node.tag.add.mytag
```

### 3.1.7 Приоритет

Правила в Synapse оцениваются в порядке приоритета. Запрашиваемое действие будет разрешено (или отклонено) на основании первого найденного соответствующего правила для этого действия. Если правило не найдено, то действие отклоняется.

Правила оцениваются от «наиболее специфичных» до «наименее специфичных». Правила оцениваются в следующем порядке:

- правила пользователя на локальном уровне (т. е. шлюз аутентификации);
- правила ролей на локальном уровне;
- правила пользователей на глобальном уровне (т. е. Cortex);
- правила ролей на глобальном уровне.

**Примечание:** Поскольку глобальные правила оцениваются после локальных правил, разрешения, предоставленные на глобальном уровне, могут «переопределять» разрешения, которые явно не запрещены на локальном уровне. Например, пользователь может разветвить представление (сделав себя администратором этого представления) и предоставить коллеге доступ «чтение» (view.read).

Если у коллеги есть разрешения «запись» (например, node.tag) на глобальном уровне, он сможет добавлять теги в разветвленном представлении (или в любом представлении, где у него есть разрешения view.read).

Если пользователь, создающий разветвление представления, также указал !node для слоя представления, коллеге будет запрещено добавлять какие-либо теги в разветвленное представление (или вносить какие-либо изменения вообще).

Роли (предоставленные пользователю) и правила (назначенные пользователю или роли) также упорядочены:

- При назначении ролей пользователю каждая новая роль добавляется в конец списка ролей, если не указано местоположение (индекс) роли.
- При назначении правил роли или пользователю каждое новое правило добавляется в конец списка правил, если не указано местоположение (индекс) правила.

Права и роли оцениваются в следующем порядке:

- Правила пользователя оцениваются в порядке от первого к последнему.
- Каждая роль, предоставленная пользователю, оценивается в порядке от первого к последнему.
- Для каждой роли правила роли оцениваются в порядке от первого к последнему.

Это означает, что те же самые правила, примененные и оцененные в разном порядке, дадут разные результаты. Простой пример:

1. Эти правила разрешат создание узлов file:bytes, но не других узлов:

```
node.add.file:bytes
!node.add
```

2. Те же правила в обратном порядке запретят создание любых узлов:

```
!node.add
node.add.file:bytes
```

### 3.1.8 Администратор

Статус администратора позволяет пользователю обходить все проверки прав доступа для области, где пользователь является администратором. Например, администратор Synapse (Cortex) может обходить все проверки разрешений Cortex (может «делать что угодно» в Cortex).

Пользователи обычно являются администраторами объектов, которые они создают. Пользователь, который разветвляет представление, является администратором для представления, которое он разветвляет, и может обходить все проверки разрешений («делать что угодно») в разветвленном представлении.

**Примечание:** Невозможно назначить роли права администратора.

### 3.1.9 Простые разрешения

Простые разрешения (сокращенно «easy perms») — это механизм, который упрощает предоставление общих наборов разрешений пользователям или ролям для определенного объекта. При использовании простых разрешений можно указать четыре уровня доступа: запрет, чтение, редактирование и администрирование. Эти уровни доступа имеют соответствующие целочисленные значения:

- Запрет = 0;
- Чтение = 1;
- Редактирование = 2;

- Администрирование = 3.

Простые разрешения применяются к определенным объектам. При наличии простых разрешений применяются следующие соглашения:

- Пользователь, создавший объект, имеет права администратора для этого объекта.
- Права администратора включают возможность предоставлять разрешения другим пользователям (включая возможность явно запретить доступ).
- Права администратора требуются для удаления объекта (т. е. разрешения на редактирование не включают в себя удаление).

**Совет:** Библиотека $lib.macro.grant — это пример того, как можно использовать простые разрешения для назначения разрешений.

### 3.1.10 Представления и слои

Данные в Cortex хранятся в одном или нескольких слоях. Слои объединяются в представления, содержащие данные, которые должны быть видны пользователям или ролям. (Стандартная установка Synapse состоит из представления по умолчанию, которое содержит один слой.)

Представления определяют данные, которые может видеть пользователь или роль — они выступают в качестве границы чтения. Предоставление разрешения view.read для представления позволяет пользователям просматривать (считывать) данные в любом из слоев представления; нет необходимости явно предоставлять доступ «чтения» к отдельным слоям.

Возможность чтения данных в представлении — «все или ничего» — не дает право пользователям видеть некоторые узлы в представлении, но не видеть другие. (Конфиденциальные данные должны храниться в своем собственном слое, а представления, содержащие этот слой, должны быть доступны пользователям или ролям, которым необходим доступ к этим данным.)

Слои определяют изменения (если таковые имеются), которые пользователь или роль могут вносить в данные в Synapse — они действуют как граница записи. В обычных условиях для записи доступен только верхний слой в представлении. Возможность записи данных в слой контролируется различными разрешениями node.*, которые определяют формы/свойства/теги/ребра, с которыми пользователь или роль может работать (создавать/изменять/удалять). Разрешения на изменение данных должны быть назначены на соответствующем слое (или глобально, если разрешения применяются ко всем доступным для записи слоям в Cortex).

## 3.2 НАЗНАЧЕНИЕ РАЗРЕШЕНИЙ

Вы назначаете (разрешаете или запрещаете) разрешения в Synapse, добавляя правила к ролям или пользователям (или удаляя правила из них). Помните, что порядок имеет значение при добавлении правил (см. п. Приоритет).

С точки зрения администратора Synapse управление разрешениями в Synapse обычно включает:

- назначение правил пользователям и ролям в Cortex;
- назначение правил пользователям и ролям для различных шлюзов аутентификации (таких как слои или представления) при необходимости;
- Назначение правил пользователям и ролям для разрешения или запрета доступа к дополнительным службам, таким как различные улучшения.

Управление разрешениями в Synapse осуществляется с помощью команд авторизации Storm.

В коммерческом пользовательском интерфейсе Optic разрешениями также можно управлять с помощью инструмента администрирования и диалоговых окон, связанных с различными объектами (такими как представления или истории).

**Совет:** Если пользователь попытается выполнить действие, на выполнение которого у него нет разрешений, Synapse вернет сообщение об ошибке проверки подлинности, в котором будет указано требуемое разрешение.

**Примечание:** В приведенных ниже описаниях и примерах предполагается, что Synapse развернут с использованием встроенного управления Synapse и аутентификации пользователей, ролей и разрешений.

### 3.2.1 РАЗРЕШЕНИЯ ПО УМОЛЧАНИЮ

Synapse включает следующие разрешения по умолчанию:

- Встроенный пользователь root имеет доступ администратора (admin=true) к Cortex.
- Встроенная роль all имеет доступ на чтение (view.read) к любому представлению, созданному с помощью worldreadable=True. Это включает представление по умолчанию.

Любые дополнительные разрешения должны быть явно предоставлены пользователям или ролям. Во всех случаях, за исключением нескольких исключительных, Synapse предполагает неявное значение по умолчанию «запретить все» в качестве окончательного правила, вычисляемого при проверке разрешений.

**Примечание:** Есть несколько исключительных случаев, когда определенное разрешение предполагает разрешение по умолчанию вместо запрета по умолчанию, но они встречаются редко. Такие случаи весьма специфичны и обычно возникают в случаях, когда

было реализовано новое разрешение. То есть действие, которое изначально не подвергалось проверке разрешений, теперь имеет такую проверку (обычно из-за необходимости явно запретить выполнение этого действия определенным пользователям или ролям).

Если бы ранее не отмеченное действие было добавлено с «запретом по умолчанию», это потенциально нарушило бы существующие развертывания Synapse, внезапно заблокировав ранее разрешенное действие (не ограничено). В этих обстоятельствах новому разрешению присваивается «разрешение по умолчанию», которое затем может быть специально отклонено при необходимости.

### 3.2.2 Доступные разрешения

Команда auth.perms.list может использоваться для отображения набора разрешений, доступных в данном Cortex. Сюда входят собственные разрешения Synapse, а также любые разрешения, связанные с другими пакетами и службами.

Пример вывода этой команды можно увидеть в разделе Разрешения Cortex. Разрешения, доступные в данном Cortex, могут различаться в зависимости от установленных служб и пакетов (например, таких как Power-Ups).

### 3.2.3 Глобальные (Cortex) разрешения

Разрешения в Synapse могут быть назначены на глобальном уровне (Cortex) или конкретному шлюзу аутентификации (см. пункт Разрешения шлюза аутентификации). Чтобы назначить разрешения для шлюза идентификации, необходимо указать его идентификатор (ident) (т. е. использовать параметр --gate для соответствующей команды Storm) при добавлении связанного правила для пользователя или роли.

Если не указать параметр аутентификации, разрешения будут глобальными и будут применяться к любому/всем экземплярам в Cortex, к которым пользователь или роль имеют доступ. Например, следующая команда Storm:

```
auth.role.addrule all node
```

предоставляет (разрешает) разрешение узла для встроенной роли all. Это позволяет любому пользователю (поскольку всем пользователям по умолчанию предоставляется роль all) выполнять любые действия с любым узлом в любом слое, который является самым верхним (доступным для записи) в любом представлении, которое может видеть пользователь.

Указание правил на глобальном уровне (Cortex) может быть достаточным для многих базовых развертываний Synapse.

**Примечание:** Следует помнить, что порядок имеет значение при добавлении правил:

- по умолчанию каждое правило добавляется в конец списка правил, назначенных пользователю или роли;

- правила оцениваются в порядке приоритета.

Чтобы изменить порядок правил, необходимо:

- использовать параметр --index с auth.user.addrule или auth.role.addrule, чтобы указать место для вставки определенного правила;
- удалить и повторно добавить правила в нужном порядке;
- использовать setRules(rules, gateiden=$lib.null) или setRules(rules, gateiden=$lib.null), чтобы заменить правила для пользователя или роли новым набором правил;
- использовать коммерческий пользовательский интерфейс Synapse (Optic), чтобы изменить порядок правил с помощью перетаскивания.

#### 3.2.3.1 Назначение разрешений

Правила разрешений (разрешить или запретить) назначаются с помощью команд auth.user.addrule и auth.role.addrule.

Например:

1. Запретим пользователю «ron» устанавливать описания тегов (установив свойство syn:tag:desc):

```
storm> auth.user.addrule ron "!node.prop.set.syn:tag:desc"
Added rule !node.prop.set.syn:tag:desc to user ron.
```

**Совет:** Правила запрета, указанные в Storm, должны быть заключены в кавычки (одинарные или двойные), поскольку они начинаются с символа ( ! ).

2. Разрешим роли «senior analysts» добавлять теги в дерево тегов определения угроз (cno.threat):

```
storm> auth.role.addrule "senior analysts" node.tag.add.cno.threat
Added rule node.tag.add.cno.threat to role senior analysts.
```

3. Запретим роли «all» удалять узлы:

```
storm> auth.role.addrule all "!node.del"
Added rule !node.del to role all.
```

4. Запретим роли «all» удалять узлы и вставим это как первое правило для роли:

```
storm> auth.role.addrule --index 0 all "!node.del"
Added rule !node.del to role all.
```

**Совет:** Помните, что отобразить пользователя или отобразить роль можно с помощью команд auth.user.show и auth.role.show.

#### 3.2.3.2 Отзыв разрешений

Правила разрешений отменяются с помощью команд auth.user.delrule и auth.role.delrule.

Например:

1. Отменим правило, запрещающее пользователю «ron» устанавливать описания тегов:

```
storm> auth.role.addrule --index 0 all "!node.del"
Added rule !node.del to role all.
```

2. Отменим правило, позволяющее «junior analysts» применять теги в дереве тегов cno.threat:

```
storm> auth.role.delrule "junior analysts" node.tag.cno.threat
Removed rule node.tag.cno.threat from role junior analysts.
```

#### 3.2.3.3 Проверка разрешений

Команда auth.user.allowed может использоваться для проверки того, имеет ли пользователь определенное разрешение (т.е. разрешено ли ему выполнять соответствующую операцию) для определенной области (т.е. глобально или для отдельного шлюза аутентификации). Если существует соответствующее разрешающее правило, команда покажет источник (т.е. правило, роль и/или связанный шлюз аутентификации), где было назначено разрешение.

**Совет:**

- У пользователя могут быть локальные разрешения (например, для доступа к определенному шлюзу аутентификации), которых у него нет глобально. Другими словами, глобальная проверка может (корректно) показать, что у пользователя нет ожидаемых глобальных разрешений, но разрешение будет отображаться как «разрешено», когда будет проверен соответствующий шлюз аутентификации.
- При проверке того, может ли пользователь просматривать (считывать) данные или манипулировать (например, разветвлять) представлением, следует проверить соответствующее представление.
- При проверке того, может ли пользователь изменять (записывать или удалять) данные, следует проверить соответствующий слой.

Например:

1. Проверим, разрешено ли пользователю «ron» применять теги в дереве тегов cno глобально:

```
storm> auth.user.allowed ron node.tag.add.cno
allowed: true - Matched role rule (node.tag.add.cno) for role
cattribution analyst.
```

2. Проверим, разрешено ли пользователю «ron» применять теги в дереве тегов cno в текущем слое:

```
    storm> auth.user.allowed --gate $lib.layer.get().iden ron
node.tag.add.cno
    allowed: true - Matched role rule (node.tag.add.cno) for role
cattribution analyst.
```

Обратите внимание, что ответ на каждую из приведенных выше команд идентичен, несмотря на то, что в первом примере выполнена глобальная проверка (без опции --gate), а во втором примере был проверен текущий слой (получен с помощью $lib.layer.get()). Ответ во втором примере показывает, что Ron может применять теги в текущем слое, поскольку у него есть глобальные разрешения на это действие — на что указывает отсутствие идентификатора в ответе. Если бы разрешения Ron’а были ограничены запрошенным шлюзом (в данном случае слоем), связанный идентификатор был бы включен в вывод команды.

3. Проверим, разрешено ли пользователю «ron» создавать ответвление текущего представления:

```
    storm> auth.user.allowed --gate $lib.view.get().iden ron view.add
    allowed: false - No matching rule found.
```

### 3.2.4 РАЗРЕШЕНИЯ ШЛЮЗА АУТЕНТИФИКАЦИИ

Чтобы назначить разрешения для шлюза аутентификации, необходимо использовать те же команды Storm, которые используются для назначения глобальных разрешений, но при добавлении или удалении правила необходимо указать полный идентификатор шлюза аутентификации (ident) (используя опцию --gate).

#### 3.2.4.1 Получение идентификатора объекта

Команды Storm view и layer можно использовать для управления представлениями и слоями соответственно. В частности, следующие команды полезны для отображения всех представлений или слоев (включая их идентификаторы) или для отображения определенного представления или слоя:

- view.list;
- view.get;
- layer.list;
- layer.get.

Например:

1. Показать все представления:

```
    storm> view.list
```

```
    View: 1e8cc6fbbae11a4c66cde03b8a17edf3 (name: default)
      Creator: 71d51feb671bf5b2d5924efdf6e29e32
      Layers:
        0fb56fd3f6a691cab6916912c48fff42: default readonly: False
```

2. Показать текущий слой:

```
    storm> layer.get
    Layer: 0fb56fd3f6a691cab6916912c48fff42 (name: default) readonly: False
creator: 71d51feb671bf5b2d5924efdf6e29e32
```

#### 3.2.4.2 Просмотр разрешений шлюза

Команда auth.gate.show используется для отображения информации о разрешениях для конкретного шлюза аутентификации (например, представления или слоя). Можно указать конкретный идентификатор для шлюза аутентификации или использовать приведенный ниже синтаксис для получения информации о текущем представлении или слое. (Просмотр информации для «текущего слоя» вернет информацию для верхнего слоя текущего представления.)

Например:

1. Отображение информации для текущего представления:

```
    storm> auth.gate.show $lib.view.get().iden
    Gate Type: view

    Auth Gate Users:
      71d51feb671bf5b2d5924efdf6e29e32 - root
        Admin: true
        Rules:

    Auth Gate Roles:
      b59a4df1148a494ea02cee09375d684f - all
        Rules:
          [0  ] - view.read
```

2. Отображение информации для текущего слоя (т.е. верхнего слоя текущего представления):

```
    storm> auth.gate.show $lib.layer.get().iden
    Gate Type: layer

    Auth Gate Users:
      71d51feb671bf5b2d5924efdf6e29e32 - root
        Admin: true
        Rules:
```

```
Auth Gate Roles:
  b59a4df1148a494ea02cee09375d684f - all
    Rules:
      [0  ] - layer.read
```

## 3.3 РЕКОМЕНДАЦИИ ПО РАЗРЕШЕНИЯМ

- Администраторам Synapse следует использовать назначенную учетную запись администратора для административных задач и отдельную учетную запись для своих пользовательских задач.
- По возможности назначайте разрешения ролям и предоставляйте роли пользователям вместо назначения разрешений пользователям напрямую.
- Создайте роль общего назначения (например, users или используйте встроенную роль all) и назначьте для этой роли основные разрешения, которые должны иметь все пользователи Synapse. Это включает «действия, которые все пользователи должны иметь возможность выполнять» (правила разрешения), а также «действия, которые всем пользователям должны быть явно запрещены» (правила запрета). Создавайте дополнительные роли по мере необходимости, чтобы разрешить (или дополнительно ограничить) определенные операции.
- Разделите данные с различными требованиями к доступу на разные слои. Предоставьте доступ к наборам данных, объединяя эти слои в представления и предоставляя ролям доступ к соответствующим представлениям.
- Возможность удалять узлы в Synapse должна быть предоставлена ограниченному числу доверенных лиц. Рекомендуется создать для этой цели специальную роль.
- Если у роли будут ограниченные права доступа, то, как правило, проще явно разрешить только эти действия; все остальное будет запрещено по умолчанию.
- Если у роли или пользователя будут обширные права доступа с некоторыми ограничениями, обычно проще сначала явно запретить ограниченные действия, а затем предоставить широкие разрешения (например, !node.del, а затем node). Поскольку правила разрешений проверяются по порядку, Synapse сначала столкнется с любыми запрещающими правилами (т.е. пользователь не может удалять узлы), блокируя запрещенное действие, а затем разрешая все, что конкретно не запрещено (т.е. пользователь может делать с узлами что-либо еще).

## 3.4 ПРИМЕРЫ РАЗРЕШЕНИЙ

Приведенные ниже примеры иллюстрируют несколько распространенных вариантов использования ролей и разрешений в Synapse. Эти наборы правил предназначены для простых иллюстраций и не обязательно иллюстрируют полностью определенные, готовые к работе наборы разрешений.

Напомним, что:

- Представления управляют доступом на чтение к хранилищу данных. Пользователи с доступом на чтение представления (view.read) могут читать все данные на всех слоях представления (т.е. для доступа на чтение не требуются дополнительные разрешения, относящиеся к конкретному слою).
- Слои управляют доступом на запись к хранилищу данных. Используйте разрешения для управления данными, которые могут быть записаны на данный слой (включая возможность объединения данных в этот слой из разветвленного представления).
- Пользователь, который может разветвлять представление, является администратором в своем разветвленном представлении.

Список доступных разрешений Cortex доступен в пункте Разрешения Cortex или может быть просмотрен в Synapse с помощью команды auth.perms.list.

### 3.4.1 Случай 1 — Предоставление общих разрешений - базовый

Базовые разрешения, представленные в таблице 2, могут быть назначены роли, чтобы позволить пользователям выполнять общие операции в Synapse.

Таблица 2 - Базовые разрешения

| Разрешение | Описание |
|---|---|
| view.read | Просмотр/чтение любых данных |
| view.add | Разветвление любого представления, которое они могут просматривать |
| node | Создание, изменение или удаление любых типов данных (узлы, свойства, границы, теги и данные узлов) в верхнем слое любого представления, которое они могут просматривать |

**Советы:**

- Роль all имеет неявный (и неотменяемый) доступ «чтения» к представлению Synapse по умолчанию. Это не то же самое, что глобальный доступ view.read. Чтобы разрешить роли all (или любой роли) видеть другие представления, необходимо явно назначить разрешение view.read (глобально или отдельным представлениям).
- Пользователи могут разветвлять (view.add) только те представления, которые они могут видеть (view.read). Если пользователям должно быть разрешено разветвлять любое представление, к которому у них есть доступ на чтение, разрешение view.add может быть назначено глобально, даже если доступ на чтение управляется для каждого представления.

### 3.4.2 Случай 2 — Предоставление общих разрешений — промежуточный вариант

Эти разрешения расширяют случай 1, но позволяют роли просматривать только определенные представления (предоставляя view.read локально отдельным представлениям).

Любые глобальные разрешения (например, node.add) будут применяться к верхнему (доступному для записи) слою любого представления, доступного роли, если только разрешения не переопределены локально.

Эти разрешения также не позволяют роли удалять узлы глобально, при этом позволяя ей удалять свойства или ребра, а также удалять теги (см. таблицу 3).

Таблица 3 - Глобальные разрешения

| Разрешение | Область | Описание |
|---|---|---|
| view.add | global | Разветвлять любое представление, которое они могут видеть (на основе view.read) |
| !node.del | global | Запретить удаление любых узлов |
| node.add | global | Создавать узлы в верхнем слое любого видимого представления |
| node.prop | global | Устанавливать, изменять или удалять свойства узла в верхнем слое любого видимого представления |
| node.edge | global | Добавлять или удалять границы в верхнем слое любого видимого представления |
| node.tag | global | Добавлять или удалять теги из узлов в верхнем слое любого видимого представления |
| view.read | local | Просматривать все данные во всех слоях определенного представления(й), где назначено правило |

### 3.4.3 Случай 3 — Создание специальной роли, которая может удалять узлы

Произвольное или неправильное удаление узлов может негативно повлиять на хранилище данных (например, оставить «дыры» в графике или уничтожить данные). Synapse требует, чтобы пользователи выполняли явную команду (delnode) для удаления узлов, поэтому это действие является осознанным выбором (в отличие от «случайного щелчка»).

Настоятельно рекомендуется создать роль, единственным разрешением которой является удаление узлов, и предоставить эту роль ограниченному числу пользователей. Для этого:

- Явно откажите в разрешении на удаление узлов (!node.del) на глобальном уровне роли общего назначения, которую используете для управления разрешениями для всех пользователей (как показано в случае 2 выше).

- Создайте специальную роль, единственным разрешением которой будет возможность удалять узлы.
    - Рекомендуется использовать название, которое внушает осторожность, например, fire ze missiles или agents of destruction, но можно просто использовать deleters.
- Назначьте правило node.del для роли (глобально или для определенных слоев).

**Совет:** Все операции удаления (будь то удаление узлов, свойств, границ или тегов) должны выполняться непосредственно в слое, где находятся данные. Как администратор любого представления, которое они разветвляют, «обычные» пользователи могут удалять данные, созданные или измененные в их разветвленном представлении.

### 3.4.4 Случай 4 — Установка ограничения на запись (создание или объединение данных)

Разрешения могут использоваться, чтобы запретить ролям:

- создавать различные типы данных непосредственно в слое/представлении;
- объединять различные типы данных в базовое представление (технически, в верхний слой представления).

Эти типы разрешений могут помочь гарантировать, что данные в «производственном» уровне останутся максимально чистыми и безошибочными. Например:

- Помогают ограничить количество опечаток, которые приводят к появлению «плохих» тегов или ребер.
- Предотвращают запись данных с конфиденциального или ограниченного слоя на слой без ограничений.

Примеры использования:

- Используйте разрешения для легких ребер, чтобы разрешить создание только определенных именованных ребер. Это может ограничить количество опечаток в названиях ребер и/или помешать пользователям создавать ребра с произвольными именами.
- Используйте разрешения для тегов или деревьев тегов, чтобы разрешить применение только определенных тегов (например, для обеспечения соблюдения правил использования тегов в организации). Например, разрешения могут гарантировать, что пользовательские «пустые» теги (#thesilence.mywork) или теги, указывающие на конфиденциальные данные (#tlp.red), не будут добавлены к «производственным» данным.
- Используйте разрешения для отдельных свойств, чтобы запретить установку определенных свойств в определенных слоях. Например, свойства классификации (такие как риск:угроза:тип) могут быть «в стадии разработки» во внутреннем

аналитическом представлении, в то время как пользователи тестируют и согласовывают соответствующие категории. Возможно, Вы захотите запретить установку (объединение) этого свойства с производственными данными до тех пор, пока классификация не будет завершена.

**Совет:** Степень, в которой Вы обеспечиваете соблюдение соглашений о данных и тегах с помощью разрешений или путем консенсуса (т.е. пользователи соглашаются «приложить все усилия» для поддержания чистоты данных), будет зависеть от организации и варианта использования. Управление разрешениями увеличивает накладные расходы, но оно того стоит, если наборы данных требуют высокой точности или качества. Накладные расходы могут принести меньшую пользу для внутренних данных или тестовых данных, где случайные ошибки оказывают минимальное влияние и могут быть «устранены» по мере необходимости.

Приведенные ниже примеры правил могут применяться глобально (пользователь с этой ролью может записывать «одобренные» данные на любой доступный для записи слой любого представления, которое он может видеть) или локально к определенным слоям.

Приведенные ниже примеры лишь иллюстрируют, как можно ограничить определенные действия по записи, и не касаются других разрешений, которые могут потребоваться пользователю/роли. Эти разрешения могут быть добавлены к существующей роли (например, к общей роли пользователя) или предоставлены через их собственную роль.

**Пример 1:**

Если разрешен ограниченный набор действий, следует просто указать изменения, которые может вносить роль. Все остальное по умолчанию неявно запрещено.

В этом примере роль со следующими разрешениями может (см. таблицу 4):

- только добавлять и удалять теги в перечисленных деревьях тегов;
- только создавать и удалять перечисленные ребра.

Таблица 4 - Назначение разрешений

| Разрешение | Описание |
|---|---|
| node.tag.add.cno | Добавить/применить теги в дереве cno (например, cno, cno.mal, cno.mal.plugx и т. д.) |
| node.tag.del.cno | Удалить все теги в дереве cno |
| node.tag.add.rep | Добавить/применить любые теги в дереве rep |
| node.tag.del.rep | Удалить все теги в дереве rep |
| node.edge.add.refs | Добавить ссылки на ребра |
| node.edge.del.refs | Удалить ссылки на ребра |
| node.edge.add.uses | Добавить пользователя ребра |
| node.edge.del.uses | Удалить пользователя ребра |

| Разрешение | Описание |
|---|---|
| node.edge.add.targets | Добавить цели ребер |
| node.edge.del.targets | Удалить цели ребер |

**Пример 2:**

Если определенные действия запрещены, следует отклонить эти изменения, а затем разрешить «все остальное».

Роли со следующими разрешениями **запрещено** (см. таблицу 5):

- создание узлов risk:threat:type:taxonomy (представляющих «категории» угроз);
- установка свойства :type для узлов risk:threat (например, указание категории классификации для конкретной угрозы);
- создание тегов в дереве tlp.

Обратите внимание, что перечисленные разрешения запрещают только действия. Чтобы роль с этими разрешениями могла вносить другие изменения (например, добавлять другие узлы или ребра), эти разрешения должны быть предоставлены после этих правил «запрета» или как часть другой роли.

Таблица 5 - Запрет действий для роли

| Разрешение | Описание |
|---|---|
| !node.add.risk:threat:type:taxonomy | Запретить создание узлов risk:threat:type:taxonomy |
| !node.prop.set.risk:threat:type | Запретить установку этого свойства (т. е. на существующих узлах risk:threat) |
| !node.tag.add.tlp | Запретить применение тегов в дереве tlp |

**Совет:** Чтобы запретить пользователям или ролям вносить какие-либо изменения в определенное представление (т. е. пользователи не могут объединять какие-либо данные в представление / записывать какие-либо данные непосредственно в самый верхний слой представления):

- Не добавляйте разрешения узлов на самый верхний слой представления (не предоставленные разрешения на запись неявно отклоняются).
- Если роли были предоставлены глобальные разрешения узлов (или аналогичные), переопределите это, явно отказав (!node) разрешение на слое, который хотите защитить.

### 3.4.5 Случай 5 — Старшие и младшие роли

Старшие роли (с большим количеством разрешений) и младшие роли (с ограниченными разрешениями) используются в различных ситуациях, например, между новичками и опытными пользователями или младшими и старшими аналитиками.

При использовании рабочего процесса «разветвления и слияния» младший пользователь может «делать все, что угодно» (как администратор) в представлении, которое он разветвляет. Это позволяет ему обогащать данные и комментировать свои оценки с помощью тегов. Но разрешения могут помешать им объединить некоторые (или все) данные и теги до тех пор, пока старший пользователь не проверит изменения. Затем старшая роль (с соответствующими разрешениями) может объединить данные от имени младшего пользователя.

Например, теги, представляющие ключевые аналитические оценки, такие как определение того, связан ли файл или индикатор с семейством вредоносных программ, или теги, представляющие кластеризацию угроз и определение их принадлежности, могут потребовать тщательного рассмотрения. Возможность объединять эти теги может быть ограничена старшими аналитиками, которые могут проверить, что младший аналитик применил их правильно.

Эти типы разрешений обычно являются **совокупными**; обычным пользователям может быть запрещено (или просто не разрешено) выполнять определенные действия, при этом дополнительные разрешения предоставляются более старшим или опытным ролям. В таблице 6 приведен пример, где все пользователи будут иметь роль обычных пользователей, а аналитикам будут предоставляться все дополнительные роли по мере накопления ими опыта.

Таблица 6 - Назначение дополнительных ролей по мере накопления опыта

| Роль | Разрешение(я) | Описание |
|---|---|---|
| users | !node.tag.cno<br>!node.tag.rep<br>node.tag | Запретить применение тегов в деревьях cno и rep (представляющих конкретные аналитические оценки), но применять другие теги |
| novice analyst | node.tag.add.rep | Новички могут применять теги в дереве rep (представляющих отчеты сторонних организаций) |
| junior analyst | node.tag.add.cno.infra | Младшие аналитики могут применять теги в дереве cno.infra (относящиеся к сетевой инфраструктуре) |
| senior analyst | node.tag.add.cno.threat<br>node.tag.add.cno.mal | Старшие аналитики могут применять теги в деревьях cno.threat и cno.mal (оценки, связанные с кластерами угроз и семействами вредоносных программ) |

**Примечание:** Из-за приоритета при предоставлении дополнительных ролей их необходимо будет добавлять (индексировать) перед ролью users, чтобы предотвратить переопределение разрешений явного запрета этой роли на новые разрешенные привилегии тегов.

### 3.4.6 Случай 6 — Специализированные роли

Для организаций с разнообразными аналитическими группами (например, где аналитики специализируются в определенных областях) или организаций, где несколько групп или отделов используют Synapse для разных целей, может быть полезно создать узкоспециализированные роли.

Например:

1. Организация со специализированной группой анализа вредоносных программ может разрешить этим специалистам применять теги, связанные с семействами вредоносных программ/кодов и экосистемами вредоносных программ.
2. Стратегические аналитики организации могут нести исключительную ответственность за определенные объекты и связанные с ними данные. Например, стратегическая группа может отвечать за исследование и создание организаций (ou:org) и связанных отраслей (ou:industry) для отслеживания виктимологии. Стратегические аналитики могут гарантировать, что эти объекты создаются в соответствии со стандартами группы и что организации отнесены к соответствующим отраслям.

Пример аналитика вредоносного ПО:

- Предполагается, что возможность применять специализированные теги, перечисленные в таблице 7, либо не предоставлена, либо явно запрещена в другом месте/для других ролей.
- Аналитикам вредоносного ПО также может быть предоставлена возможность удалять теги, перечисленные в таблице 7, с соответствующими разрешениями node.tag.del.

Таблица 7 - Разрешения аналитика вредоносного ПО

| Разрешение | Описание |
|---|---|
| node.tag.add.cno.code | Применять теги cno.code (обозначающие конкретные образцы семейств кода - например, cno.code.plugx) |
| node.tag.add.cno.mal | Применять теги cno.mal (обозначающие компоненты экосистем семейств вредоносных программ/кодов, такие как связанные дропперы или C2 — например, cno.mal.plugx) |
| node.tag.add.cno.rel | Применять теги cno.rel (обозначающие компоненты, которые могут рассматриваться как часть экосистемы вредоносных программ, но не являются по своей сути вредоносными, например, cno.rel.plugx) |

Пример стратегического аналитика (см. таблицу 8):

- Предполагается, что возможность создавать эти узлы/устанавливать эти свойства либо не предоставлена, либо явно запрещена в другом месте/другим ролям.
- Стратегическим аналитикам может быть дополнительно предоставлена возможность удалять соответствующие узлы/свойства с соответствующими разрешениями node.del или node.prop.del.
- В зависимости от того, как назначаются разрешения, стоит иметь в виду, что роли, которые не могут создавать узлы, могут иметь возможность устанавливать или изменять свойства узла, пока узел уже существует. При необходимости эту возможность можно ограничить с помощью дополнительных правил node.prop.set.

Таблица 8 - Разрешения стратегического аналитика

| Разрешение | Описание |
|---|---|
| node.add.ou:org | Создавать организационные узлы |
| node.prop.set.ou:org:industries | Назначать организации одной или нескольким отраслям |
| node.add.ou:industry | Создавать отраслевые узлы |

## 3.5 Разрешения Cortex

Ниже приведен список разрешений Cortex, которые могут быть предоставлены пользователю или роли. Если указан шлюз, отличный от cortex, разрешение будет проверено на соотвествие по конкретному экземпляру шлюза, и если соответствие не найдено, оно будет проверено по глобальным правилам.

```
storm> auth.perms.list
auth.role.set.name
    Permits a user to change the name of a role.
    gate: cortex
    default: false

auth.role.set.rules
    Permits a user to modify rules of a role.
    gate: cortex
    default: false
```

```
auth.self.set.apikey
    Permits a user to manage their API keys.
    gate: cortex
    default: true

auth.self.set.email
    Permits a user to change their own email address.
    gate: cortex
    default: true

auth.self.set.name
    Permits a user to change their own username.
    gate: cortex
    default: true

auth.self.set.passwd
    Permits a user to change their own password.
    gate: cortex
    default: true

auth.user.get.profile.<name>
    Permits a user to retrieve their profile information.
    gate: cortex
    default: false
    example: auth.user.get.profile.fullname

auth.user.grant
    Controls granting roles to a user.
    gate: cortex
    default: false

auth.user.pop.profile.<name>
```

```
        Permits a user to remove profile information.
        gate: cortex
        default: false
        example: auth.user.pop.profile.fullname

    auth.user.revoke
        Controls revoking roles from a user.
        gate: cortex
        default: false

    auth.user.set.admin
        Controls setting/removing a user's admin status.
        gate: cortex
        default: false

    auth.user.set.apikey
        Permits a user to manage API keys for other users. USE WITH CAUTUON!
        gate: cortex
        default: false

    auth.user.set.archived
        Controls archiving/unarchiving a user account.
        gate: cortex
        default: false

    auth.user.set.email
        Controls changing a user's email address.
        gate: cortex
        default: false

    auth.user.set.locked
        Controls locking/unlocking a user account.
```

```
        gate: cortex
        default: false

    auth.user.set.passwd
        Controls changing a user password.
        gate: cortex
        default: false

    auth.user.set.profile.<name>
        Permits a user to set profile information.
        gate: cortex
        default: false
        example: auth.user.set.profile.fullname

    auth.user.set.rules
        Controls adding rules to a user.
        gate: cortex
        default: false

    axon.del
        Controls the ability to remove a file from the Axon.
        gate: cortex
        default: false

    axon.get
        Controls the ability to retrieve a file from the Axon.
        gate: cortex
        default: false

    axon.has
        Controls the ability to check if the Axon contains a file.
        gate: cortex
```

```
        default: false

    axon.upload
        Controls the ability to upload a file to the Axon.
        gate: cortex
        default: false

    backup.del
        Permits a user to delete an existing backup.
        gate: cortex
        default: false

    backup.list
        Permits a user to list existing backups.
        gate: cortex
        default: false

    backup.run
        Permits a user to create a backup.
        gate: cortex
        default: false

    cron.add
        Permits a user to create a cron job.
        gate: view
        default: false

    cron.del
        Permits a user to remove a cron job.
        gate: cronjob
        default: false
```

```
cron.get
    Permits a user to list cron jobs.
    gate: cronjob
    default: false

cron.kill
    Controls the ability to terminate a running cron job.
    gate: cronjob
    default: false

cron.set
    Permits a user to modify/move a cron job.
    gate: cronjob
    default: false

cron.set.creator
    Permits a user to modify the creator property of a cron job.
    gate: cortex
    default: false

globals
    Used to control all operations for global variables.
    gate: cortex
    default: false

globals.get
    Used to control read access to all global variables.
    gate: cortex
    default: false

globals.get.<name>
    Used to control read access to a specific global variable.
```

```
    gate: cortex
    default: false

globals.pop
    Used to control delete access to all global variables.
    gate: cortex
    default: false

globals.pop.<name>
    Used to control delete access to a specific global variable.
    gate: cortex
    default: false

globals.set
    Used to control edit access to all global variables.
    gate: cortex
    default: false

globals.set.<name>
    Used to control edit access to a specific global variable.
    gate: cortex
    default: false

model.form.add
    Controls access to adding extended model forms.
    gate: cortex
    default: false

model.form.add.<form>
    Controls access to adding specific extended model forms.
    gate: cortex
    default: false
```

```
        example: model.form.add._foo:bar

    model.form.del
        Controls access to deleting extended model forms.
        gate: cortex
        default: false

    model.form.del.<form>
        Controls access to deleting specific extended model forms.
        gate: cortex
        default: false
        example: model.form.del._foo:bar

    model.prop.add
        Controls access to adding extended model properties.
        gate: cortex
        default: false

    model.prop.add.<form>
        Controls access to adding specific extended model properties.
        gate: cortex
        default: false
        example: model.prop.add._foo:bar

    model.prop.del
        Controls access to deleting extended model properties and values.
        gate: cortex
        default: false

    model.prop.del.<form>
         Controls access to deleting specific extended model properties and
values.
```

```
        gate: cortex
        default: false
        example: model.prop.del._foo:bar

    model.tagprop.add
        Controls access to adding extended model tag properties and values.
        gate: cortex
        default: false

    model.tagprop.del
        Controls access to deleting extended model tag properties and values.
        gate: cortex
        default: false

    model.type.add
        Controls access to adding extended model types.
        gate: cortex
        default: false

    model.type.add.<type>
        Controls access to adding specific extended model types.
        gate: cortex
        default: false
        example: model.type.add._foo:bar

    model.type.del
        Controls access to deleting extended model types.
        gate: cortex
        default: false

    model.type.del.<type>
        Controls access to deleting specific extended model types.
```

```
        gate: cortex
        default: false
        example: model.type.del._foo:bar

    model.univ.add
        Controls access to adding extended model universal properties.
        gate: cortex
        default: false

    model.univ.del
         Controls access to deleting extended model universal properties and
values.
        gate: cortex
        default: false

    node
        Controls all node edits in a layer.
        gate: layer
        default: false

    node.add
        Controls adding any form of node in a layer.
        gate: layer
        default: false

    node.add.<form>
        Controls adding a specific form of node in a layer.
        gate: layer
        default: false
        example: node.add.inet:ipv4

    node.data.pop
```

*Permits a user to remove node data in a given layer.*

*gate: layer*

*default: false*

*node.data.pop.<key>*

*Permits a user to remove node data in a given layer for a specific key.*

*gate: layer*

*default: false*

*example: node.data.pop.hehe*

*node.data.set*

*Permits a user to set node data in a given layer.*

*gate: layer*

*default: false*

*node.data.set.<key>*

*Permits a user to set node data in a given layer for a specific key.*

*gate: layer*

*default: false*

*example: node.data.set.hehe*

*node.del*

*Controls removing any form of node in a layer.*

*gate: layer*

*default: false*

*node.del.<form>*

*Controls removing a specific form of node in a layer.*

*gate: layer*

*default: false*

*node.prop*

*Controls editing any prop on any node in the layer.*

*gate: layer*

*default: false*

*node.prop.del*

*Controls removing any prop on any node in a layer.*

*gate: layer*

*default: false*

*node.prop.del.<form>*

*Controls removing any property from a form of node in a layer.*

*gate: layer*

*default: false*

*example: node.prop.del.inet:ipv4*

*node.prop.del.<form>.<prop>*

*Controls removing a specific property from a form of node in a layer.*

*gate: layer*

*default: false*

*example: node.prop.del.inet:ipv4.asn*

*node.prop.set*

*Controls setting any prop on any node in a layer.*

*gate: layer*

*default: false*

*node.prop.set.<form>*

*Controls setting any property on a form of node in a layer.*

*gate: layer*

*default: false*

*example: node.prop.set.inet:ipv4*

*node.prop.set.<form>.<prop>*

*Controls setting a specific property on a form of node in a layer.*

*gate: layer*

*default: false*

*example: node.prop.set.inet:ipv4.asn*

*node.tag*

*Controls editing any tag on any node in a layer.*

*gate: layer*

*default: false*

*node.tag.add*

*Controls adding any tag on any node in a layer.*

*gate: layer*

*default: false*

*node.tag.add.<tag...>*

*Controls adding a specific tag on any node in a layer.*

*gate: layer*

*default: false*

*example: node.tag.add.cno.mal.redtree*

*node.tag.del*

*Controls removing any tag on any node in a layer.*

*gate: layer*

*default: false*

*node.tag.del.<tag...>*

*Controls removing a specific tag on any node in a layer.*

*gate: layer*

*default: false*

```
        example: node.tag.del.cno.mal.redtree

    pkg.add
        Controls access to adding storm packages.
        gate: cortex
        default: false

    pkg.del
        Controls access to deleting storm packages.
        gate: cortex
        default: false

    service.add
        Controls the ability to add a Storm Service to the Cortex.
        gate: cortex
        default: false

    service.del
        Controls the ability to delete a Storm Service from the Cortex
        gate: cortex
        default: false

    service.get
        Controls the ability to get the Service object for any Storm Service.
        gate: cortex
        default: false

    service.get.<iden>
        Controls the ability to get the Service object for a Storm Service by
iden.
        gate: cortex
        default: false
```

```
    service.list
        Controls the ability to list all available Storm Services and their
service definitions.
        gate: cortex
        default: false

    storm.asroot.cmd.<cmdname>
        Controls running storm commands requiring root privileges.
        gate: cortex
        default: false
        example: storm.asroot.cmd.movetag

    storm.asroot.mod.<modname>
        Controls importing modules requiring root privileges.
        gate: cortex
        default: false
        example: storm.asroot.cmd.synapse-misp.privsep

    storm.graph.add
        Controls access to add a storm graph.
        gate: cortex
        default: true

    storm.inet.imap.connect
        Controls connecting to external servers via imap.
        gate: cortex
        default: false

    storm.inet.smtp.send
        Controls sending SMTP messages to external servers.
        gate: cortex
```

```
        default: false

    storm.lib.auth.roles.add
        Controls the ability to add a role to the system. USE WITH CAUTION!
        gate: cortex
        default: false

    storm.lib.auth.roles.del
          Controls the ability to remove a role from the system. USE WITH
CAUTION!
        gate: cortex
        default: false

    storm.lib.auth.users.add
        Controls the ability to add a user to the system. USE WITH CAUTION!
        gate: cortex
        default: false

    storm.lib.auth.users.del
          Controls the ability to remove a user from the system. USE WITH
CAUTION!
        gate: cortex
        default: false

    storm.lib.axon.del
        Controls the ability to remove a file from the Axon.
        gate: cortex
        default: false

    storm.lib.axon.get
        Controls the ability to retrieve a file from the Axon.
        gate: cortex
        default: false
```

*storm.lib.axon.has*

*Controls the ability to check if the Axon contains a file.*

*gate: cortex*

*default: false*

*storm.lib.axon.wget*

*Controls the ability to retrieve a file from URL and store it in the Axon.*

*gate: cortex*

*default: false*

*storm.lib.axon.wput*

*Controls the ability to push a file from the Axon to a URL.*

*gate: cortex*

*default: false*

*storm.lib.cortex.httpapi.add*

*Controls the ability to add a new Extended HTTP API on the Cortex.*

*gate: cortex*

*default: false*

*storm.lib.cortex.httpapi.del*

*Controls the ability to delete an Extended HTTP API on the Cortex.*

*gate: cortex*

*default: false*

*storm.lib.cortex.httpapi.get*

*Controls the ability to get or list Extended HTTP APIs on the Cortex.*

*gate: cortex*

*default: false*

```
storm.lib.cortex.httpapi.set
    Controls the ability to modify an Extended HTTP API on the Cortex.
    gate: cortex
    default: false

storm.lib.inet.http.proxy
    Permits a user to specify the proxy used with `$lib.inet.http` APIs.
    gate: cortex
    default: false

storm.lib.log.debug
    Controls the ability to log a debug level message.
    gate: cortex
    default: false

storm.lib.log.error
    Controls the ability to log a error level message.
    gate: cortex
    default: false

storm.lib.log.info
    Controls the ability to log a info level message.
    gate: cortex
    default: false

storm.lib.log.warning
    Controls the ability to log a warning level message.
    gate: cortex
    default: false

storm.lib.stix.export.maxsize
      Controls the ability to specify a STIX export bundle maxsize of
```

```
greater than 10,000.
        gate: cortex
        default: false

    storm.lib.telepath.open
        Controls the ability to open an arbitrary telepath URL. USE WITH
CAUTION.
        gate: cortex
        default: false

    storm.lib.telepath.open.<scheme>
        Controls the ability to open a telepath URL with a specific URI
scheme. USE WITH CAUTION.
        gate: cortex
        default: false

    storm.macro.add
        Controls access to add a storm macro.
        gate: cortex
        default: true

    storm.macro.admin
        Controls access to edit/set/delete a storm macro.
        gate: cortex
        default: false

    storm.macro.edit
        Controls access to edit a storm macro.
        gate: cortex
        default: false

    task.del
        Controls access to terminate other users tasks.
```

```
        gate: cortex
        default: false

    task.get
        Controls access to view other users tasks.
        gate: cortex
        default: false

    trigger.add
        Controls adding triggers.
        gate: cortex
        default: false

    trigger.del
        Controls deleting triggers.
        gate: view
        default: false

    trigger.get
        Controls listing/retrieving triggers.
        gate: trigger
        default: false

    trigger.set
        Controls enabling, disabling, and modifying the query of a trigger.
        gate: view
        default: false

    trigger.set.<property>
        Controls modifying specific trigger properties.
        gate: view
        default: false
```

```
trigger.set.doc
    Controls modifying the doc property of triggers.
    gate: trigger
    default: false

trigger.set.name
    Controls modifying the name property of triggers.
    gate: trigger
    default: false

trigger.set.user
    Controls modifying the user property of triggers.
    gate: cortex
    default: false

view
    Controls all view permissions.
    gate: cortex
    default: false

view.add
    Controls access to add a new view including forks.
    gate: cortex
    default: false

view.del
    Controls access to delete a view.
    gate: view
    default: false

view.fork
```

```
        Controls access to fork a view.
        gate: view
        default: true

    view.read
        Controls read access to view.
        gate: view
        default: false

    view.set.<setting>
        Controls access to change view settings.
        gate: view
        default: false
        example: view.set.name
```

## 3.6 РАЗРЕШЕНИЯ OPTIC

Коммерческим клиентам Synapse, использующим Optic UI, может потребоваться явно предоставить пользователям или ролям разрешение на использование некоторых инструментов пользовательского интерфейса (например, Spotlight).

Информацию о развертывании Optic см. в руководстве по развертыванию Optic.

Информацию о разрешениях Optic и других функциях см. в руководстве Optic DevOps.

**Совет:** Не нужно явно предоставлять разрешения самому Optic. Если Вы создаете и управляете пользователями и ролями Synapse («Cortex») через Optic, у них по умолчанию есть разрешение на доступ к Optic.

## 3.7 РАЗРЕШЕНИЯ НА POWER-UP

У Synapse Power-Ups есть свои собственные наборы разрешений, которые должны быть предоставлены пользователям или ролям, чтобы они могли использовать Power-Up и любые связанные команды Storm. Конкретные разрешения описаны в разделе Admin Guide документации Power-Up для отдельного Power-Up.

**Совет:** Хотя большинство предоставляемых Vertex Power-Ups являются частью коммерческого предложения Synapse, следующие Rapid Power-Ups также доступны для использования с общедоступной версией Synapse (с открытым исходным кодом):

- Synapse-MISP;
- Synapse-MITRE-ATT&CK;
- Synapse-PSL (список общедоступных суффиксов FQDN);
- Synapse-TOR.

## 3.8 Разрешения среды выполнения Storm

Когда пользователь запускает запрос Storm в интерактивном режиме (например, в командной строке Storm или через панель запросов Optic) или выполняет действие в пользовательском интерфейсе Optic (например, доступ к пункту меню), запрос или действие выполняется с разрешениями пользователя на основе применимых разрешений пользователя и разрешений роли, а также текущей области действия запроса или действия.

В некоторых случаях при выполнении в среде выполнения Storm используются другие разрешения, которые могут потребовать дополнительных действий.

### 3.8.1 Автоматизация

Synapse включает возможность автоматизировать задачи на основе Storm с помощью триггеров, заданий cron и/или макросов. Разрешения по-разному влияют на все эти элементы, в том числе:

- кто может создавать автоматизацию или управлять ей (например, по умолчанию любой пользователь может создать макрос, но для создания триггеров или заданий cron требуются явные разрешения);
- кто запускает определенную часть автоматизации (например, макросы запускаются от имени пользователя, который их выполняет, но триггеры и задания cron запускаются от имени пользователя, который их создал).

Подробное обсуждение автоматизации в Synapse (включая вопросы разрешений) см. в разделе Справочник Storm — Автоматизация Руководства пользователя Synapse.

### 3.8.2 Power-Ups

Power-Ups реализуют пакеты Storm и сервисы Storm для предоставления Synapse дополнительных функций. Power-Ups могут предоставляться The Vertex Project (в виде бесплатных или коммерческих предложений). Организации также могут разрабатывать собственные Power-Ups.

Power-Ups обычно устанавливают команды Storm, чтобы пользователи могли использовать дополнительные возможности Power-Up. В некоторых случаях Power-Ups может потребоваться доступ к конфиденциальным данным (например, ключам API или аналогичным учетным данным) или выполнение действий (например, при добавлении узлов или применении тегов), которые некоторым пользователям не разрешено выполнять самостоятельно.

Power-Ups могут использовать разделение привилегий («privsep»), чтобы ограниченное подмножество возможностей Power-Up могло работать с повышенными привилегиями, если это необходимо, а остальная часть кода работала от имени пользователя, вызывающего Power-Up.

Дополнительные сведения см. в разделе «Быстрая разработка Power-Up» Руководства разработчика Synapse.

**Примечание:** Администраторы Synapse обычно отвечают только за то, чтобы соответствующие пользователи и роли могли использовать или запускать отдельные Power-Ups (см. Разрешения на Power-Up). Хотя администраторам Synapse следует помнить о разделении привилегий в Power-Up как о лучшей практике, реализация разделения привилегий остается за разработчиками Power-Up.

# 4 ДОБАВЛЕНИЕ РАСШИРЕННЫХ ЭЛЕМЕНТОВ МОДЕЛИ

Модель данных Synapse в Cortex может быть расширена пользовательскими формами или свойствами с помощью библиотеки расширения модели Storm ($lib.model.ext). Имена форм и свойств расширенной модели должны начинаться со знака подчеркивания (_), чтобы избежать потенциальных конфликтов имен со встроенными элементами модели.

**Примечание:** Используемые элементы расширенной модели (имеющие узлы, использующие расширенные формы или свойства), нельзя удалить, пока не будут удалены все экземпляры этого элемента расширенной модели. Другими словами, перед удалением расширенных форм необходимо сначала удалить узлы, созданные с помощью этой расширенной формы, а перед удалением расширенных свойств необходимо удалить значение свойства из узлов с этим расширенным свойством.

## 4.1 РАСШИРЕННЫЕ ФОРМЫ

При добавлении формы $lib.model.ext.addForm принимает следующие аргументы:

- formname — Имя формы; должно начинаться со знака подчеркивания (_) и содержать хотя бы одно двоеточие (:).
- basetype — Тип модели данных Synapse для формы.
- typeopts — Словарь опций конкретного типа.
- typeinfo — Словарь информационных значений для формы.

Чтобы добавить новую форму с именем _foocorp:name, содержащую строковые значения, которые будут нормализованы до нижнего регистра, с удалением пробелов в начале/конце:

```
$typeopts = ({'lower': $lib.true, 'strip': $lib.true})
$typeinfo = ({'doc': 'Foocorp name.'})


$lib.model.ext.addForm(_foocorp:name, str, $typeopts, $typeinfo)
```

Если форма больше не используется и в Cortex нет узлов этой формы, ее можно удалить с помощью:

```
$lib.model.ext.delForm(_foocorp:name)
```

## 4.2 РАСШИРЕННЫЕ СВОЙСТВА

При добавлении свойств $lib.model.ext.addFormProp принимает следующие аргументы:

- formname — Название формы, в которую добавляется свойство; может быть встроенной или расширенной типовой формой.
- propname — Относительное название свойства; должно начинаться со знака подчеркивания (_).
- typedef — Запись (type, typeopts), которая определяет тип для свойства.
- propinfo — Словарь информационных значений для свойства.

Чтобы добавить свойство с именем _score в форму _foocorp:name, содержащую целочисленные значения от 0 до 100:

```
    $typeopts = ({'min': 0, 'max': 100})
    $propinfo = ({'doc': 'Score for this name.'})

    $lib.model.ext.addFormProp(_foocorp:name, _score, (int, $typeopts), $propinfo)
```

Чтобы добавить свойство с именем _aliases в форму _foocorp:name, содержащую уникальный массив значений ou:name:

```
    $typeopts = ({'type': 'ou:name', 'uniq': $lib.true})
    $propinfo = ({'doc': 'Aliases for this name.'})

    $lib.model.ext.addFormProp(_foocorp:name, _aliases, (array, $typeopts), $propinfo)
```

Свойства также можно добавлять к существующим формам, например, чтобы добавить свойство с именем _classification к inet:fqdn, которое должно содержать строку из предопределенного набора значений:

```
    $typeopts = ({'enums': 'unknown,benign,malicious'})
    $propinfo = ({'doc': 'Classification for this FQDN.'})

    $lib.model.ext.addFormProp(inet:fqdn, _classification, (str, $typeopts), $propinfo)
```

## 4.3 Расширенные универсальные свойства

Подобно $lib.model.ext.addFormProp, $lib.model.ext.addUnivProp принимает те же аргументы propname, typedef и propinfo, но применяется ко всем формам.

# 5 УПРАВЛЕНИЕ УСТАРЕВАНИЕМ МОДЕЛЕЙ

По мере роста и развития модели данных Synapse элементы модели (типы, формы и свойства) могут стать устаревшими, и больше не должны использоваться для моделирования новых данных. Команды Storm model.deprecated можно использовать для подготовки к возможному удалению устаревших элементов модели.

## 5.1 БЛОКИРОВКА УСТАРЕВШИХ ЭЛЕМЕНТОВ МОДЕЛИ

Команда model.deprecated.lock изменяет статус блокировки устаревших элементов модели. Заблокированные элементы модели по-прежнему можно просматривать или удалять, но добавлять их больше нельзя. Попытка добавить заблокированный элемент модели вызовет ошибку IsDeprLocked. Команда model.deprecated.locks может использоваться для отображения текущего статуса блокировки всех устаревших элементов модели.

Например:

1. Заблокируем свойство ps:person:img:

```
storm> model.deprecated.lock ps:person:img
Locking: ps:person:img
```

2. Разблокируем свойство ps:person:img:

```
storm> model.deprecated.lock --unlock ps:person:img
Unlocking: ps:person:img
```

3. Заблокируем все устаревшие элементы модели:

```
storm> model.deprecated.lock *
Locking all deprecated model elements.
```

## 5.2 ПРОВЕРКА УСТАРЕВШИХ ЭЛЕМЕНТОВ МОДЕЛИ

Команда model.deprecated.check проверяет состояние блокировки и наличие устаревших элементов модели в Cortex. Предупреждения будут выдаваться для любых устаревших элементов модели, которые разблокированы или все еще используются в Cortex. После устранения всех предупреждений Cortex будет готов к будущим обновлениям модели.

# 6 НАСТРОЙКА ЗЕРКАЛЬНОГО СЛОЯ

**Примечание:** Зеркальные слои устарели в версии 2.x и будут удалены в версии 3.0.0. Команды layer.pull.add и layer.push.add можно использовать для настройки потокового редактирования в/из слоев в отдельном Cortex.

После настройки зеркального слоя ему необходимо будет передавать всю историю событий из вышестоящего слоя. Во время этого процесса слой будет доступен для чтения, но записи будут зависать из-за необходимости ожидания обратной записи. Cortex может быть настроен на зеркалирование слоя из удаленного Cortex, который будет синхронизировать все изменения из удаленного слоя и использовать поддержку обратной записи для упрощения изменений, исходящих из нижестоящего слоя. Зеркальный слой будет точной копией слоя в удаленной системе, включая всю историю изменений, и будет разрешать только те изменения, которые сначала будут отправлены в вышестоящий слой.

При настройке зеркального слоя можно выбрать зеркалирование из удаленного слоя или из верхнего слоя удаленного представления. Если выбрать зеркалирование из верхнего слоя удаленного представления, это представление будет иметь возможность запускать триггеры и применять ограничения модели к изменениям, предоставляемым зеркальным слоем.

Чтобы указать удаленный слой в качестве исходного, используйте URL-адрес Telepath, который содержит общий объект */layer/<layeriden>, например:

```
aha://cortex.loop.vertex.link/*/layer/8ea600d1732f2c4ef593120b3226dea3
```

Чтобы указать удаленное представление, используйте общий объект */view/<viewiden>, например:

```
aha://cortex.loop.vertex.link/*/view/8ea600d1732f2c4ef593120b3226dea3
```

При указании параметра --mirror в команде layer.add или в определении слоя, предоставленном API $lib.layer.add() Storm, URL-адрес telepath проверяться не будет. Это позволяет настроить удаленный слой или представление, которые еще не подготовлены или в данный момент находятся в автономном режиме.

**Примечание:** Чтобы разрешить доступ на запись, URL-адрес telepath должен разрешать администратору доступ к удаленному Cortex из-за возможности создания источников редактирования. URL-адрес telepath может использовать псевдонимы или сертификаты на стороне клиента TLS для предотвращения раскрытия учетных данных.

После настройки зеркального слоя ему потребуется передать всю историю событий из вышестоящего слоя. Во время этого процесса слой будет доступен для чтения, но записи будут зависать из-за необходимости ожидания полной обратной записи, чтобы гарантировать, что изменения будут сразу видны, как и в обычном слое. Во время этого процесса можно отслеживать ход выполнения, вызывая API getMirrorStatus() для объекта layer в среде выполнения Storm.

# ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ

| *№ версии* | *Дата изменения* | *Автор* | *Состав изменений* |
|---|---|---|---|
| 1.0 | 07.03.2025 | Антонова Ю.В. | Первоначальная версия документа. |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |