CKAN Practical Report

Implementing and Demonstrating CKAN Features on a Custom Domain

---

Submitted by:

- Chuang Ma [u7703248]
- Diao Fu [u7722376]
- Anbo Wu [u7706346]
- Qifeng Zheng [u7775861]

Australian National University (ANU)

Introduction

This report is to detail our hands-on experience in implementing and demonstrating CKAN (Comprehensive Knowledge Archive Network) features on a custom domain.

CKAN is an open-source data management system designed to facilitate the storage, sharing, and discovery of datasets. In this project, we explored various aspects of CKAN, including backend and frontend development, API exploration, and participation in the CKAN community interaction. The goal is to gain practical insights into CKAN's architecture, functionality, and extensibility while applying theoretical knowledge in a real-world scenario.

We started the project by registering a domain and setting up a cloud-based server on Oracle Cloud Infrastructure (OCI) to host our CKAN instance. This report covers the domain registration process, cloud setup, and subsequent configuration to ensure the CKAN environment was fully operational. Following the installation and setup phase, the focus shifted to configuring CKAN’s core functionalities, such as creating organizations, datasets, and users, while customizing the platform's appearance and behavior.

One of the cores for this project was experimenting with CKAN's extensibility. By enabling and configuring various CKAN extensions, we enhanced the system’s capabilities, such as providing advanced data visualization and interactive tables. We worked with multiple extensions like datatables_view, geo_view, image_view, choroplethmap, and more to understand their impact on CKAN's flexibility and usability. Alongside these, we explored the CKAN API to programmatically manage datasets and resources, conducting CRUD (Create, Read, Update, Delete) operations using both the ckanapi library and HTTP requests via the requests module.

In the later stages, we also engaged with the CKAN community by contributing a pull request to resolve an issue encountered during development.

This report not only covers the technical steps we took but also reflects on the challenges faced, the solutions implemented, and the lessons learned throughout the process.

Part 1. Domain Registration & Compute Instance Creation

Domain Registration on GoDaddy [1]

The first step in setting up our CKAN demonstration environment involved registering a domain. We used GoDaddy for this purpose The domain we registered was **ckandemo.site**, which serves as the base URL for our CKAN instance.

- Search for Domain Availability: We checked for the availability of "ckandemo.site" using GoDaddy's search feature.
- Purchase Domain: Once confirmed that the domain was available, we followed the steps to purchase it.

Creating a Free Compute Instance on OCI [2]

This step was to set up the server that would host our CKAN instance. We chose Oracle Cloud Infrastructure (OCI) for this purpose because of its free tier offering, which provides ample compute resources for running a CKAN demo.

- Create a Free Compute Instance(OCPU count: 4, Memory: 24GB): We chose Ubuntu 22.04 LTS as the operating system because it is a stable and CKAN supported Linux distribution ideal for hosting CKAN services.
- Instance Configuration: We configured the networking settings, ensuring that the instance had public internet access and assigned a public IP. We enabled firewall rules (security lists) to allow HTTP (the default port 80 was changed to 8888) traffic, necessary for web access to CKAN.

- DNS Setup with GoDaddy: Once the instance was up and running, we updated our DNS A record on GoDaddy to point to the public IP of the OCI instance. This step linked the domain "ckandemo.site" to the newly created server.

Reflection

1. The domain registration process was efficient and simple. GoDaddy’s intuitive dashboard made it easy to configure DNS settings. However, a challenge faced was understanding the nuances of DNS configuration, especially with regards to correctly setting A and CNAME records to point the domain to the upcoming cloud server.
2. While OCI provides a powerful and cost-effective platform for hosting, proper configuration of security settings is crucial to ensure both functionality and protection against unauthorized access. To enhance security, We changed the default port 80 to 8888 to reduce vulnerability to potential attacks. This step was essential in ensuring that the CKAN instance operates safely while minimizing exposure to common security risks associated with widely known default ports.
3. For future improvements, it might be helpful to document every step meticulously, especially for team members who might not be familiar with cloud setups.

Part 2. The Process of Installing and Configuring

This section details the process of installing and configuring CKAN. The installation was performed using the source installation method, which provides greater flexibility and control over the setup process. The decision to install CKAN from source was made to gain a deeper understanding of the system's architecture and to have more control over the installation process. Please refer to our **CKAN installation record document [3]** for detailed installation steps.

Detailed Installation Process

1.Preparing the Environment

- Process:

  The first step in the installation process involved preparing the server environment. This included updating the system packages and installing the necessary dependencies. The process highlighted the importance of having a clean, up-to-date system as a foundation for the CKAN installation.

- Reflections:

  The preparatory stage is crucial for a smooth installation process. It's important to carefully review and understand each dependency before installation. Creating a checklist of required packages and their versions could be helpful for future installations.

2. Installing CKAN from Source

- Process:

  Installing CKAN from source can be done by cloning the CKAN repository to a local device and setting up a Python virtual environment. Ensuring the correct version of Python and other dependencies were installed. Managing the Python virtual environment effectively. Careful reading of the CKAN

documentation to ensure all prerequisites were met. Creating a separate virtual environment for CKAN to avoid conflicts with other Python applications.

- Reflections:

  The source installation method provides valuable insights into CKAN's structure. It's crucial to keep track of the virtual environment activation to ensure all commands are run in the correct context. Documenting each step of the process is invaluable for troubleshooting and future reference.

3. Setting up the Database

- Process:

  CKAN uses PostgreSQL as its database. Setting up the database involved creating a new PostgreSQL user and database for CKAN. Configure the database connection string correctly in CKAN's configuration file. Double-checking all credentials and connection details before proceeding.

- Reflections:

  Understanding basic PostgreSQL administration is crucial for a successful CKAN installation. It's important to use strong, unique passwords for database users in production environments. Keeping database credentials secure and separate from the main configuration file could enhance security.

4. Configuring Solr for Search Functionality

- Process:

  Solr is used by CKAN to provide fast and advanced search functionality. Setting up Solr involved installing the software, creating a Solr core for CKAN, and configuring CKAN to use this core. Ensuring compatibility between the Solr version and CKAN. Carefully following the version recommendations in the CKAN documentation.

- Reflections:

The Solr setup process emphasizes the modular nature of CKAN's architecture. Understanding basic Solr concepts helps in troubleshooting and optimizing search functionality.

5. Initial CKAN Configuration

- Process:

  Configuring CKAN involved modifying the main configuration file to set various options, including database connection strings, site URL, and plugin settings. Understanding the numerous configuration options available. Balancing between default settings and customized options for specific needs. Carefully reading through the CKAN documentation to understand each configuration option. Starting with a minimal configuration and gradually adding options as needed.

- Reflections:

  The flexibility of CKAN's configuration allows for fine-tuned customization. It's important to document any deviations from the default configuration for future reference.

6. Setting up the Web Server

- Process:

  The installation process involved setting up Nginx as a reverse proxy and using uWSGI as the application server for CKAN. Configuring Nginx to properly proxy requests to the CKAN application. Setting up uWSGI to work with CKAN and Nginx. Using Supervisor to manage the uWSGI process, ensuring it starts automatically.

- Reflections:

  The web server setup emphasizes the importance of understanding web server concepts and configurations. Using Supervisor for process management adds an extra layer of reliability to the CKAN installation.

Consider implementing HTTPS for enhanced security in production environments.

8. Enabling and Configuring Extensions

- Process:

  CKAN's functionality can be extended through various plugins. The installation process involved enabling and configuring several extensions to enhance CKAN's capabilities. Ensuring compatibility between different extensions and the CKAN core. Resolving dependencies between extensions. Carefully reading each extension's documentation and requirements. Installing extensions one at a time and testing functionality before moving to the next. Troubleshooting conflicts by reviewing log files and adjusting configurations as needed.

  Extensions Installed: *"dsaudit", "activity", "viewhelpers", "excelforms", "tabledesigner", "datastore", "xloader", "datatables_view", "text_view", "image_view", "video_view", "audio_view", "webpage_view", "stats", "pdf_view", "resource_proxy", "geo_view", "navigablemap", "choroplethmap", "linechart", "barchart", "piechart", "basicgrid"*

- Reflections:

  The extensibility of CKAN is one of its strongest features, allowing for customization to specific needs. It's important to balance the desire for additional functionality with the potential complexity and maintenance overhead of multiple extensions. Regularly updating extensions along with the CKAN core is crucial for maintaining security and compatibility.

Challenges and Solutions

Throughout the installation process, several challenges were encountered. Here's a summary of the major issues and their solutions:

1. Module Import Errors:

   Several extensions faced issues with missing modules or directories. This was often due to incomplete installations or mismatches between the installed location and the expected location.

   Solution: Manually copying the missing files or directories from the source to the expected location. This highlighted the importance of verifying the installation process for each extension.

2. CORS Issues:

   The Pie chart and Line chart views occasionally failed to load due to CORS (Cross-Origin Resource Sharing) errors.

   Solution: Modifying the CKAN configuration to allow all origins for CORS. In a production environment, this should be more tightly controlled for security reasons.

3. File Upload Limitations:

   Errors occurred when attempting to upload larger data files.

   Solution: Adjusting the maximum file size settings in the CKAN configuration. This emphasized the need to consider resource limitations and adjust configurations accordingly.

4. Search Functionality Issues:

   The simple search functionality was not returning results as expected.

   Solution: Temporarily disabling the multilingual plugins. This highlighted the potential conflicts that can arise between different functionalities and the need for careful testing when enabling multiple features.

5. Artifact File Locations:

   Several extensions encountered issues with missing files or directories, particularly related to i18n (internationalization) and static assets.

   Solution: Manually copying the required files from the source directories to the expected locations. This underscored the importance of understanding the file structure expectations of CKAN and its extensions.

Comments

Advantages

- Flexible and Extensible Architecture

  One of CKAN's most notable strengths is its highly flexible and extensible architecture. The core CKAN system provides a robust foundation for data management and the extensive plugin ecosystem allows users to customize to meet specific organizational needs. This modular approach enables organizations to start with a basic setup and gradually add functionality as required, without the need for a complete overhaul of the system. That allowing organizations to benefit from collective expertise and development efforts. During the installation process, it became obvious that CKAN's architecture is designed with scalability in mind. The separation of concerns between different components (e.g., the main application, database, search engine, and caching system) allows for independent scaling and optimization of each component based on specific usage patterns and requirements.

- Active Community and Ecosystem

  The installation process highlighted the pros of CKAN's community and its ecosystem. The availability of extensive documentation, active forums, and a wide range of community-contributed extensions demonstrate the vibrancy of the CKAN ecosystem. This active community not only provides valuable support during installation and troubleshooting but also contributes to the ongoing evolution and improvement of the platform.

Opportunities for Enhancement

- Enhanced Security for Database Connections

  Currently, CKAN's configuration file requires database connection details, including usernames and passwords, to be stored in plain text. This approach, while functional, poses potential security risks.

  Suggestion: Implement support for more secure authentication methods for database connections, such as certificate-based authentication.

- Improved Plugin Configuration Management

  The current method of configuring plugins in CKAN involves listing all enabled plugins in a single line within the configuration file. This approach, while simple, can become unwieldy as the number of plugins grows and doesn't clearly communicate plugin dependencies.

  Suggestion: Implement a more structured and verbose plugin configuration system. This could involve: Creating a dedicated section in the configuration file for plugin management. Allowing each plugin to be configured on a separate line, with options to specify dependencies and load order. Implementing a hierarchical configuration structure that visually represents plugin dependencies.

  For example:

  ```
  [ckan.plugins]
  plugin1 = enabled
  plugin2 = enabled
      depends = plugin1
  plugin3 = enabled
      depends = plugin1, plugin2
  ```

  This approach would make the configuration more readable, easier to manage, and less prone to errors arising from incorrect plugin load orders.

- Streamlined Extension Management

  While CKAN's extensibility is a strength, managing multiple extensions, their dependencies, and potential conflicts can be challenging.

Suggestion: Develop a built-in extension management system. Features could include: A command-line tool for installing, updating, and removing extensions. Automatic dependency resolution for extensions. Version compatibility checking between extensions and the CKAN core. A web-based interface for managing extensions, integrated into the CKAN admin panel. This would simplify the process of managing CKAN's functionality and help administrators maintain a stable and up-to-date system.

- Automated Deployment and Configuration Management

  While the manual installation process provides valuable insights into CKAN's architecture, it can be time-consuming and prone to human error, especially for complex deployments.

  Suggestion: Develop official support for automated deployment methods. Such as developing a CKAN-specific deployment tool that guides users through the installation process, automating steps where possible and providing clear instructions for manual interventions when needed.

Reflections

1. Complexity and Learning Curve

Installing CKAN from source is a complex but rewarding process that provides deep insights into the architecture and capabilities of this powerful data management system. The challenges encountered during the installation process, from resolving dependencies to configuring various components, all contribute to a comprehensive understanding of CKAN's working mechanism. It requires a deep understanding of various technologies including Python, PostgreSQL, Solr, Redis, and web servers.

2. Documentation Importance

The CKAN documentation proved to be an invaluable resource throughout the installation process. However, there were instances where the documentation could be more detailed or up-to-date, particularly regarding some extensions.

3. Modular Architecture

The modular nature of CKAN, with its core functionality extended by a rich ecosystem of extensions, offers great flexibility but also introduces complexity in terms of management and compatibility. Each component (database, search engine, caching system, etc.) needs to be correctly configured and integrated. This underscores the importance of careful planning, thorough testing, and ongoing maintenance.

4. Extension Ecosystem

The wide range of available extensions is a strength of CKAN, allowing for customization to specific needs. However, managing multiple extensions can be challenging, particularly ensuring compatibility and resolving conflicts.

5. Security Considerations

The installation process highlighted various security considerations, from database user permissions to web server configurations. In a production environment, additional security measures would be necessary.

6. Community Importance

The CKAN community, through forums and issue trackers, proved to be a valuable resource for troubleshooting. This highlights the strength of open-source projects with active communities.

7. Moving forward

Staying engaged with the CKAN community, keeping abreast of new developments in the CKAN ecosystem, and continuously refining and optimizing the installation will be key to maximizing the value derived from this powerful tool.

Part 3. Backend Management

This section explains how to manage the backend of CKAN (Comprehensive Knowledge Archive Network), focusing on changing its look and features. CKAN is an open-source system for storing, sharing, and organizing data. To change how it looks and works, you often need to work with the backend directly. Please check our **CKAN backend management demo guide [4]** for detailed steps.

Detailed Customization Process & Insights

1.**Adding Static Files to CKAN**

The first step was creating a new folder to store custom static files. This is important for organizations that want to add their own images, stylesheets, or other static content to their CKAN system. Being able to add custom static files gives a lot of flexibility for adapting CKAN to meet the needs of different organizations.

2.Configuring CKAN to Recognize Additional Static File Locations

After creating the folder, the next step was changing CKAN's configuration so that it can use this new folder. This was done by adding the new folder path to CKAN's `extra_public_paths` setting. This setting allows CKAN to serve extra static files. Changing these configuration files directly on the server requires being careful with how the files are written and making sure the file permissions are correct.

3.Preparing and Adding a Featured Image

To add a custom featured image, we created the image file and uploaded it to the new static file folder on the server. This process of creating, uploading, and linking to the image can be made easier with a more automated workflow.

4.Modifying Template Snippets

In this step, we edited CKAN's HTML templates to link to the new featured image and change the layout of the homepage. Being able to change templates gives lots of control over how CKAN looks. However, it does require knowing HTML and Jinja2 templates, which might be hard for some users.

Setting a Custom Favicon

To set a custom favicon, we created the favicon file, uploaded it to the server, and

changed CKAN’s configuration to link to the new favicon. Like the featured image process, this could also be made easier with a simpler workflow.

Applying Changes

Finally, after making all the changes, we restarted the CKAN system to apply them. Restarting CKAN can be disruptive, especially in a live environment, because users might not be able to access the system during the restart.

Comments

Advantages

- Extensibility: CKAN's design makes it easy to expand, so organizations can change it to fit their needs.
- Fine-grained Control: Administrators can change templates and settings, giving them control over how the system works and looks.
- Transparent Architecture: CKAN’s open backend helps administrators understand how the system works, making it easier to fix problems and improve it.
- Community Support: The CKAN community is very active and helps by sharing tips and solving tricky customization problems.

Recommendations for Improvement

- Web-based Customization Interface

  Developing a comprehensive web-based interface for common customization tasks could significantly improve usability without sacrificing flexibility.

- Visual Theme Editor

  Implementing a visual theme editor could make it easier for non-technical users to customize CKAN's appearance.

- Dynamic Update System

  Developing a system for applying certain changes without requiring full system restarts could improve maintainability and reduce downtime.

- Version Control Integration

  Integrating version control capabilities for configuration files and templates could improve change management and enable easier rollbacks.

Reflections

Flexibility vs. Usability

CKAN’s backend management gives lots of flexibility, but it can make things harder for non-technical users. Changing templates and settings allows for many custom options, but it needs technical skills that not all organizations have.

Learning Curve

Using CKAN’s backend means you need to understand its design and file setup well. This can be tough for new administrators to learn, but it also helps them get a better idea of how the system works.

Version Control Considerations

The direct editing of configuration files and templates highlights the importance of version control. Implementing a system for tracking changes and enabling rollbacks could significantly improve the management process.

Balancing Customization and Standardization

While extensive customization options are valuable, they can lead to divergence from the standard CKAN experience. This balance between customization and standardization is a common challenge in open-source projects.

Contribution Opportunities

The challenges encountered during the customization process highlight areas where contributions back to the CKAN project could be valuable, such as developing more user-friendly customization tools.

Part 4. Frontend Management

This section details our experience in managing the frontend aspects of a CKAN instance, exploring various features, customizations, and extensions. Please refer to our **CKAN Frontend management demo documentation [5]** for specific demonstration steps.

Detailed Process

Customizing CKAN's Look and Feel

- Site Configuration

  One of the first tasks in setting up our CKAN instance was customizing its appearance to align with our project's identity. CKAN provides a user-friendly interface for administrators to modify key elements of the site's look and feel.

- Site Title and About Section

  We began by updating the site title to "CKAN Demo-8715" to clearly identify our instance. The “about” section was crucial in conveying the purpose of our project. This description effectively communicates our project's context and objectives, helping visitors understand the purpose of our CKAN instance.

- Site Logo

  For the site logo, we opted to use a modified version of the official CKAN logo (https://ckan.org/static/img/ckan-dpg.svg). This choice maintains a connection to the CKAN brand while allowing for some customization to suit our project's identity.

- Intro Text

  We crafted an introductory text to welcome visitors and provide a brief overview of the site's purpose.

- Custom CSS

To further customize the appearance, we implemented a simple CSS modification. This change alters the background color of the header (masthead) to a teal shade, giving our instance a distinct visual identity.

User Management and Organization Creation

- User Registration Process

  To simulate a real-world scenario, we created multiple user accounts with varying roles and affiliations. The registration process in CKAN is straightforward, requiring basic information such as username, full name, email, and an optional profile picture. We created several users.

- Organization Creation

  CKAN's organization feature is crucial for grouping related datasets and managing user permissions. We created several organizations to represent different data sources and governmental bodies. Using the user created in the previous step, this user will automatically become the administrator of the organization.

Dataset Management

- Adding New Datasets

  A key feature of CKAN is adding and managing datasets. We tried adding different datasets to our organizations to act like a real-world data portal. The process of adding a dataset includes several steps:

  Entering basic info like title, description, and tags.

  Uploading or linking to data resources.

  Choosing visibility and access settings.

Adding custom fields as needed. CKAN’s strength is allowing unlimited custom fields, so organizations can capture extra info that might be important for their specific needs.

- Data Resources
  CKAN lets you add multiple resources to each dataset in formats like CSV, XML, and JSON. We added different formats to show how CKAN handles various data types.

User Management and Organization Creation

User Registration Process

To mimic a real-world situation, we made multiple user accounts with different roles and groups. The registration process in CKAN is simple, needing only basic info like username, full name, email, and an optional profile picture. We created several users.

Organization Creation

CKAN’s organization feature is important for grouping related datasets and controlling user permissions. We created several organizations to represent different data sources and government groups. The user from the previous step automatically became the admin of the organization.

Data Preview and Visualization

One of CKAN's most powerful features is its ability to provide data previews and visualizations directly within the platform. This functionality enhances data accessibility and understanding for users. We explored various view plugins to showcase different types of data effectively.

- DataTables View

  The DataTables view plugin provides an interactive table interface for structured data stored in the DataStore (CSV format). To enable this view, we

added the datatables_view plugin to the CKAN configuration file. Then, we created a new view for our dataset and configured it.

The resulting view provided several powerful features:

- ❖ Multi-column sorting: Users can sort data by multiple columns to organize information according to specific needs.
- ❖ Search and filter: Built-in search and filtering options allow users to quickly find and display data that meets specific criteria.
- ❖ Paginated views: Large datasets are automatically split into pages, improving performance and making data easier to browse.
- ❖ Column visibility: Users can choose which columns to show or hide, customizing the view to focus on relevant information.

- Text View

  The Text view plugin is useful for displaying plain text data directly in the browser. We enabled the text_view plugin in the CKAN configuration file and created a new text view for a suitable dataset. This view is particularly useful for displaying files in XML, JSON, or plain text formats with syntax highlighting. It provides a quick and easy way for users to inspect the contents of text-based data files without needing to download them.

- Image View

  The Image view plugin allows for the direct display of image files within CKAN. We enabled the image_view plugin and created an image view for a dataset containing visual data. This view is particularly useful for datasets that include maps, graphs, or other visual content, allowing users to quickly view and assess image files without needing external software.

- Video View

  To showcase multimedia content, we enabled the video_view plugin and created a video view for a dataset containing video files. The Video view supports three video formats: MP4, WebM, and OGG. It provides basic video playback functionality within the CKAN interface.

- Audio View

  For datasets containing audio files, we enabled the audio_view plugin and created an audio view. This view relies on the HTML5 <audio> tag and supports three audio formats: MP3, WAV, and OGG.

- Web Page View

  The Web Page view plugin allows for the embedding of external web content within CKAN. We enabled the webpage_view plugin and created a web page view for a dataset that linked to an external resource. This view adds an <iframe> tag to embed the resource URL, which is particularly useful for showcasing external resources such as dashboards, reports, or other web-based content without requiring users to leave the CKAN platform.

- PDF View

  To handle PDF documents, we installed the ckanext-pdfview extension and enabled the pdf_view plugin. This extension provides a view plugin for PDF files using an HTML object tag, making it convenient for users to view PDF documents without needing to download them.

- Geo View

  For geospatial data, we installed the ckanext-geoview extension and enabled the geo_view plugin. This extension contains view plugins to display various geospatial files and services in CKAN . The Geo View plugin supports a wide range of formats and services, including Web Map Service (WMS), Web Feature Service (WFS), GeoJSON, GML, KML, ArcGIS REST API, and Google Fusion Tables. In our case, we showcased the handling of GeoJSON data in an OpenLayers environment.

- Choropleth Map

  To create more advanced geographical visualizations, we installed the ckanext-mapviews extension and enabled the choroplethmap and navigablemap plugins. This extension adds robust mapping capabilities to

CKAN, allowing for the creation of interactive maps, including navigable and choropleth maps.

For our demonstration, we created a new dataset: "The Internet Usage per 100 People in 2012 All Across the World". To create a choropleth map, we needed two things: the data we wanted to plot, and a GeoJSON defining the geographical regions we'd like to plot it on. The data itself needed to be in a resource inside the DataStore, and the map needed to be in the same domain as CKAN itself. We set up the map by mapping the 'Country Code' field in our dataset to the 'WB_A3' field in the GeoJSON file, which allowed us to identify each country correctly.

- Basic Charts

  To provide more options for data visualization, we installed the ckanext-basiccharts extension and enabled the linechart, barchart, piechart, and basicgrid plugins. We then created various chart types for our "Electric Vehicle Population Size History By County" dataset.

  - ❖ Pie Chart: We created a pie chart to visualize the distribution of electric vehicles across different counties. The setup process involved selecting the appropriate columns for the chart's sectors and values.
  - ❖ Line Chart: For time-series data in the dataset, we created a line chart to show the growth of electric vehicle adoption over time. This involved setting up the x and y axes and grouping the data appropriately.
  - ❖ Basic Grid: We also set up a Basic Grid view, which provides a simple tabular representation of the data.

Table Designer and Data Dictionary

To enhance data management capabilities, we enabled the tabledesigner plugin and set up the DataStore. We also installed the ckanext-excelforms extension to support Excel-based data entry.

- Creating a Table Designer Resource

We created a new dataset called "Table Designer Test" to demonstrate this feature. The Table Designer allows for the creation of structured data resources directly within CKAN.

- Creating Fields with the Data Dictionary

  The Data Dictionary feature allows for the definition of fields and their properties. We accessed this feature through two paths: either directly from the resource page or through the "Manage" dropdown menu. For each field, we could define various properties such as the field type (text, number, date, etc.), whether it's required, and any constraints (like minimum or maximum values for numeric fields).

- Creating and Updating Rows with the Web Form

  Table Designer offers a web form for interactively creating or updating individual rows. This feature is particularly useful for small datasets or for making quick updates to existing data.

- Creating or Updating Rows with ckanext-excelforms

  For bulk data entry or updates, we used the ckanext-excelforms extension. This allows users to download an Excel template based on the defined schema, fill it with data, and then upload it back to CKAN.

  The process involves:

  - Downloading the Excel template
  - Filling in the data in Excel
  - Uploading the completed Excel file
  - Submitting the data or checking for errors before submission

Search Functionality

CKAN's search functionality is a critical feature for users to find relevant datasets. The platform supports two search modes: simple search and advanced search.

- Simple Search

Simple search is triggered when the search terms entered contain no colon (":"). It uses Solr's DisMax Query Parser, which is designed to be robust and handle most queries without error.

For example, searching for "demo +100" will look for all datasets containing the word "demo" and filter only those also matching "100", as the "+" makes it a mandatory term.

- Advanced Search

  Advanced search is activated when the search expression contains at least one colon (":"). This mode allows for more precise queries, including field-specific searches.

  For instance, searching for "title:Electric" will look for all datasets containing the word "Electric" specifically in their title field.

Comments

Customizing CKAN's Look and Feel

- The process of customizing CKAN's appearance through the admin interface was straightforward and effective. The ability to modify key elements like the site title, “about” section, logo, and intro text without delving into backend code makes it accessible for administrators who may not have extensive technical expertise.
- We noted some limitations in the customization options available through the web interface. For instance, changing the featured image and setting the favicon still require manual backend modifications. Integrating these options

into the web-based interface would streamline the customization process, making it more intuitive for site administrators to manage all visual aspects without needing to access the server backend.

- While the custom CSS option is powerful, it could be enhanced with a visual editor or predefined themes to make it more accessible to non-technical users. This would allow for more comprehensive visual customization without requiring CSS knowledge.

User Management and Organization Creation

- The process of creating users and organizations in CKAN is generally smooth and user-friendly. The ability to quickly assign users as administrators for specific organizations ensures that roles and responsibilities are clearly defined from the outset.
- We identified some areas for improvement:

  The absence of an "About" field during the initial user registration process is a missed opportunity. Adding this field to the registration form would encourage users to provide more context about their role or expertise immediately, enhancing the overall user profiling capabilities of the system.

  After creating the necessary organizations, we recommend setting ***ckan.auth.user_create_organizations = false*** in the configuration file. This disables the ability for regular users to create organizations, ensuring that only authorized administrators can manage organizational entities. This is important for maintaining control over the structure of the platform, especially in larger or more sensitive deployments.

Dataset Management

- Creating and managing datasets in CKAN through the Create Dataset page is generally user-friendly and intuitive. The ability to add multiple resources in various formats to a single dataset is particularly useful, allowing for comprehensive data presentation.
- We identified several areas for potential improvement:

Custom Fields Persistence: When adding custom fields to a dataset, it would be beneficial to have an option to make these fields persistent within the organization without requiring global configuration changes by the site administrator. This would provide greater autonomy and flexibility for each organization's data management practices.

File Upload Feedback: When uploading data files on the Add Data page, especially for larger files, the inclusion of a progress bar would greatly improve the user experience. This visual feedback on the upload status would make CKAN more accommodating to handling large datasets, reducing user uncertainty during the upload process.

Data Preview and Visualization

- DataTables View

  The DataTables view is a robust solution for presenting structured data. Its sorting, filtering, and search capabilities make it particularly useful for large datasets. However, it's important to note that this plugin requires the data to be structured and stored in the DataStore, which may require additional setup for some datasets.

- Text View

  While the Text view is simple and effective for small to medium-sized files, we noticed some performance issues when handling larger files. The view tends to slow down significantly, and in some cases, the page becomes unresponsive or the file fails to load altogether. To improve this, it would be helpful to implement a system that loads only a portion of the file initially, with an option to continue loading more data as needed. This approach would enhance the user experience and prevent timeouts or failures when dealing with large datasets.

- Image View

  The Image view functions well for its intended purpose, providing a straightforward way to display image content. However, we noticed that the

Filters and Add Filter buttons on the page do not seem to function as expected.

- Video View

  The Video view works well for basic video playback. We identified an area for improvement. Currently, users have to provide a URL for the poster image (thumbnail) for the video. Adding a poster image upload function directly in the view configuration would streamline the process, allowing users to upload thumbnail images from their local file system without relying on external image hosting.

- Audio View

  The Audio view provides basic audio playback functionality, which is sufficient for most use cases. However, we noticed that the colors of the played and unplayed portions of the progress bar are quite similar, making it difficult to distinguish between them during playback. Adjusting the color contrast between these sections would significantly improve the user experience, making it easier to track playback progress at a glance.

- Web Page View

  The Web Page view is powerful for integrating external content, it's important to use it cautiously. We recommend not activating this plugin unless you trust the URL sources, as it could potentially be used to embed malicious content. It's not recommended to enable this view type on instances where all users can create datasets.

  If the target website has disabled cross-site iframe embedding, this view will not work, rendering the content inaccessible within CKAN. It would be helpful if CKAN could provide a fallback option or at least a clear error message in such cases.

- PDF View

The PDF view is a valuable addition for document-based datasets, allowing for quick previews and enhanced interaction. It's worth noting that if PDF files are hosted externally (on a different server than CKAN), the resource_proxy plugin must also be enabled. This dependency could be made clearer in the setup process to ensure smooth implementation.

- Geo View

  The Geo View plugin greatly enhances CKAN's ability to handle and display geospatial data. Its support for multiple formats makes it versatile for various geospatial data types. However, the setup process can be complex, especially for users unfamiliar with geospatial data standards.

- Choropleth Map

  The choropleth mapping capability is a powerful feature for visualizing geographically distributed data. However, we encountered a couple of issues during our implementation:

  - Despite enabling the ***choroplethmap*** plugin in the configuration file, there was no "Choropleth Map" option available in the chart types list on the resource page.
  - When creating a navigable map, the Value Attribute field was missing from the page, preventing the full setup of the map.

  These issues may require checking the plugin's integration with the CKAN version being used or reviewing the plugin's setup documentation for possible misconfigurations. Improving the robustness of these plugins and providing clearer error messages or troubleshooting guides would greatly enhance the user experience when working with these advanced visualization features.

- Basic Charts

  The basic charts extension provides a good range of visualization options that are relatively easy to set up and can effectively visualize various data types, including numeric, text, and date. The grouping feature is particularly useful for comparing data across different categories or time periods.

We noticed some areas for improvement:

- In the line chart, the x and y axis labels are stacked on top of each other, making them difficult to read. Improving the label placement would enhance the readability of the charts.
- The Basic Grid functionality appears to be redundant, as it can be fully replaced by the existing Table View in CKAN. This duplication of features may lead to confusion for users and unnecessary complexity in the system.
- There was an issue with the Bar Chart option not appearing when creating a view, despite barchart being enabled in the configuration file. This inconsistency needs to be addressed to ensure all chart types are available as expected.

Table Designer and Data Dictionary

The Table Designer and Data Dictionary features, combined with the Excel forms capability, provide a powerful toolset for structured data management within CKAN. Some key strengths include:

- The ability to use the CKAN DataStore as the primary data source, which allows for more efficient querying and data manipulation.
- Enforcement of validation rules based on the defined schema, ensuring data integrity.
- Support for updating rows without reloading all data, which is efficient for large datasets.
- Seamless interaction with ckanext-excelforms for bulk uploads, which is crucial for managing large datasets.

We also identified some limitations and areas for improvement:

- When maintaining the Data Dictionary, fields cannot be reordered, which can make schema management less easy. Adding the ability to reorder fields would improve the user experience.

- We encountered a bug when adding maximum or minimum value constraints to integer fields, causing issues during data validation. The error messages were:

  Copypsycopg2.errors.UndefinedFunction: operator does not exist: text < bigint
  LINE 1: NEW."ID" < '1'::int8
  psycopg2.errors.UndefinedFunction operator does not exist: text > bigint
  LINE 1: NEW."ID" > '99999'::int8

  This bug needs to be addressed to ensure proper validation of numeric fields.

Search Functionality

The search functionality in CKAN is powerful and versatile. Some key strengths include:

- The simple search mode is user-friendly and can handle a wide range of queries without requiring users to learn complex syntax.
- Support for phrase searches and modifiers like "+" and "-" in simple search mode provides additional flexibility.
- Advanced search offers powerful features like wildcards, proximity matching, and field-specific searches, making it a valuable tool for users needing precision.

There are some areas where the search functionality could be improved:

- Providing users with syntax hints, particularly when using advanced search, would enhance usability. This could include a list of valid fields that can be searched.
- We noticed that when the multi-language plugin is enabled, the simple search may fail to return results in some cases. This issue needs to be investigated and resolved to ensure consistent search functionality across all CKAN configurations.
- Integrating features like autocomplete or search suggestions could help users formulate more effective queries.

Reflections

Throughout this project, we gained a deeper understanding of CKAN, particularly in its customization, data management, and user interface functionalities. Here are the key takeaways:

CKAN’s Flexibility

We learned how to modify CKAN’s appearance and functionality, giving us control over the design and how users interact with the system. Customizations, such as changing the site title, logo, intro text, and adding custom CSS, provided hands-on experience in making a CKAN instance reflect the identity of our project.

User and Organization Management

By creating users with different roles, we learned how to manage permissions efficiently. This gave us the knowledge to simulate real-world scenarios in which different users have varying levels of access and responsibility.

Data Management

We learned how to add multiple data resources to a dataset, preview data in various formats, and configure different types of views, such as tables, text, images, videos, and geospatial maps. This understanding of dataset management is essential for creating a data portal that provides users with easy access to valuable information.

Data Visualization

We realized the importance of visuals when dealing with large datasets because they help make data more digestible for end-users. Understanding how to configure views and set up plugins for visualization is crucial for making CKAN useful for non-technical users.

DataStore and Table Designer

We learned how to use the Table Designer to create structured datasets directly within CKAN. This allowed us to define custom fields and manage the data interactively, both through the web form and via Excel uploads. These features provide a robust way to handle data inside CKAN, ensuring the data is well-organized and easily accessible.

Search Functionality

By understanding how Solr powers the search engine in CKAN, we learned how to optimize search queries for better results. This aspect of CKAN is fundamental for users who rely on finding specific datasets within a large catalog.

Part 5. CKAN API Implementation

This section details our practice with the CKAN API. We explored various API functionalities using both the ckanapi library and direct HTTP requests with the requests module. Please refer to the details in our Jupyter Notebook file.[6]

Process

Environment Setup

We began with setting up our environment. This involved installing the ckanapi library and importing necessary modules. We defined our CKAN instance URL and obtained an API token for authentication.

API Exploration

We systematically explored different CKAN API functions:

- Retrieving Data. We implemented functions to:
    - ❖ List organizations for a user
    - ❖ List packages (datasets)
    - ❖ Show package details
- Creating Data. We created functions to:
    - ❖ Create new packages
    - ❖ Create resources within packages
    - ❖ Create data table in datastores
- Updating Data. We developed functions to:
    - ❖ Update packages
    - ❖ Update resources
    - ❖ Patch packages
    - ❖ Patch resources
    - ❖ Upsert data in datastores

- Deleting Data. We implemented functions to:
    - ❖ Delete resources
    - ❖ Delete packages

Comparison of Methods

For each API function, we implemented two versions:

- Using the CKAN API library
- Using direct HTTP requests with the requests module

This allowed us to compare the two approaches and understand their pros and cons.

Error Handling

We implemented basic error handling in our functions, including catching specific CKAN API errors and general exceptions.

Testing

We tested each function with sample data. This helped us verify the correct functioning of our implementations and understand the API's behavior.

Comments

Strengths

- Comprehensive API Coverage:

  The CKAN API demonstrates impressive breadth, encompassing a wide range of functionalities. It provides robust support for data publication, metadata management, user and organization administration, search capabilities, and resource handling.

- Flexible Versioning System:

CKAN's API versioning system stands out for its user-friendly approach. By allowing developers to specify the desired API version in the URL, it strikes a balance between backward compatibility and feature evolution. This flexibility is crucial for maintaining long-term projects while enabling access to newer functionalities.

- Wide-range of language choices:

  Despite CKAN's core code is written in Python, the API's design allows for access from any programming language capable of making HTTP requests. It allows a wide range of languages to choose, which significantly enhances CKAN's integration potential, making it an option for various development environments and existing systems.

- Strong Data Management:

  The API provides strong control over data management operations. Functions for creating, updating, patching, and deleting both packages and resources offer developers sufficient control over data structures and content.

- Support for Partial Updates:

  The inclusion of patch methods for both packages and resources is a valuable feature. This allows for partial updates to data, which can be more efficient than full updates, especially when dealing with complex data structures.

Weaknesses

- Inconsistent Response Handling:

  The API's use of different HTTP status codes for various response states introduces inconsistencies in error handling. This can complicate client-side error management and lead to potential robustness issues in applications built on top of CKAN.

- Compulsory Update Method Behavior:

The current behavior of the update method, which deletes parameters not explicitly provided in the update request, can lead to unintended data loss. This approach may cause issues, especially when updating complex data structures or when client applications don't have a complete view of the existing data.

- Insufficient Permission Documentation:

  The API documentation lacks detailed information about required permissions for each API function. This gap in documentation can lead to potential security problems if developers expose sensitive operations by accident due to misunderstanding of permission requirements.

Suggestions for Improvement

- Standardize Response Handling:

  Implement a consistent response structure across all API endpoints, regardless of success or failure status. This could involve always returning a 200 OK status and including success/failure indicators and error details in the response body.

- Refine Update Method Behavior:

  Introduce an option in the update method to only modify specified fields while leaving others unchanged. This would prevent unintended data loss and provide more predictable behavior.

- Enhance Permission Documentation:

  Develop comprehensive documentation detailing the specific permissions required for each API function. This should include information on user roles, scopes, and any conditional permissions.

Reflections

Balancing Flexibility and Complexity

One of the primary challenges we encountered was navigating the balance between the API's flexibility and its resulting complexity. While CKAN's API offers great flexibility, this comes at the cost of a steeper learning curve. We had to spend much time in understanding CKAN's data model and API conventions to use it effectively.

API Design Principles

Our practice with the CKAN API provided valuable insights into effective API design principles. We learned the importance of balancing flexibility with consistency, and how design choices impact the developer experience. The CKAN API's comprehensive coverage demonstrated how a well-designed API can serve as a powerful interface for complex systems.

Authentication and Security

Working with the CKAN API reinforced the importance of robust authentication mechanisms. We gained practical experience in implementing token-based authentication and learned to appreciate the security implications of different API operations. This experience highlighted the critical nature of proper permission management in applications focusing on data.

Error Handling Strategies

The inconsistencies in CKAN's error handling prompted us to reflect on best practices for error management in API design. We learned the importance of consistent error reporting and how it affects the robustness and maintainability of client applications.

Versioning Strategies

CKAN's approach to API versioning provided insights into strategies for maintaining backward compatibility while allowing for API evolution. We learned how version

management in APIs can significantly impact long-term project maintenance and client application stability.

Real-world Application of RESTful Principles

Our practice provided a concrete application of RESTful API principles in a complex, real-world scenario. We gained a deeper understanding of how these principles translate into practical implementation and the challenges involved in adhering to them.

Part 6. Participating in the CKAN Community

A Pull request

For more details, please refer to our 'Participating in the CKAN community' [7] documentation.

Detailed Process

- Identifying the Issue

  While working on the CKAN demo project, our team encountered a startup error when adding the "basicgrid" extension. The error occurred due to an assertion failure in the *ckan/ckan/lib/jinja_extensions.py* file. The assertion, *assert searchpath and current_path*, was triggered because both *searchpath* and *current_path* variables were *empty* or *None*.

  After analyzing the code, we discovered that the issue came from the configuration item *'extra_template_paths'* in the *ckanext-basiccharts/ckanext/basiccharts/plugin.py* file. The value of this item ended with an extraneous comma, which caused the subsequent conversion to a list in *ckan/ckan/config/environment.py* to result in an empty value within *config['computed_template_paths']*.

- Proposing a Solution

  To address this issue, we proposed a simple solution: filter out any *empty* or *None* values from the *template_paths* list before assigning it to *config['computed_template_paths']*. Specifically, we added the following line of code in ckan/ckan/config/environment.py, before the existing assignment:*template_paths = list(filter(None, template_paths)) .* This change ensured that any *empty* or *None* values were removed from the *template_paths* list, preventing the assertion failure.

- Submitting a Pull Request

With the proposed solution in hand, we proceeded to submit a pull request to the CKAN project on GitHub. The pull request included a detailed description of the issue, the root cause analysis, and the suggested fix.

- Responding to Community Feedback

  After submitting the pull request, the CKAN community provided valuable feedback. They pointed out that modifying *'extra_template_paths'* was not the recommended approach for adding template paths in CKAN. Instead, the preferred method is to use the *add_template_directory* function, as per the CKAN guidelines. The community also suggested that, rather than modifying the code, it would be better to generate an error and prevent CKAN from starting in the event of configuration issues. This approach would help identify and resolve problems more effectively, rather than silently working around them.

- Closing the Pull Request

  Based on the feedback received from the CKAN community, we concluded that our proposed solution, while technically valid, did not align with the recommended best practices for the project. Therefore, we decided to close the pull request.

Comments

- The CKAN community's feedback and guidance were invaluable in this process. They highlighted the importance of following the project's established best practices and guidelines, even when a more straightforward technical solution is available.
- The community's suggestion to generate an error and prevent CKAN from starting in case of configuration issues was particularly insightful. This approach prioritizes the robustness and maintainability of the system, rather than working around problems. It encourages developers to address the root cause of issues and improve the overall quality of the codebase.

Reflections

- Participating in the CKAN community through this pull request was a valuable learning experience. It emphasized the significance of understanding the broader context and best practices within an open-source project, rather than focusing solely on the immediate technical problem.
- The analysis and solving process highlighted the importance of thorough root cause analysis.
- This interaction reinforced the need to carefully consider the implications of proposed changes, even if they appear to be simple fixes.
- The community's suggestion to generate an error and prevent CKAN from starting in case of configuration issues was a valuable lesson.

Overall, participating in this CKAN community pull request was a transformative experience. It taught us the importance of aligning with project-specific guidelines, considering the broader implications of our changes, and prioritizing the long-term maintainability and reliability of the system. These lessons will undoubtedly give us many ideas about our future contributions to open-source projects and our approach to software.

Conclusion

This CKAN implementation project provides valuable insights into managing and customizing an open-source data management platform. Throughout the project, we learned how to effectively set up and configure CKAN, manage datasets, and enable extensions to create a robust data portal. The flexibility of CKAN's architecture is evident, as it allows extensive customization, making it possible to be adaptable to a wide range of organizational needs. However, the other side of great flexibility may be complexity which occurs when dealing with backend configurations and plugin management.

One of the key takeaways from this project is the importance of thorough documentation and community support. This kind of transparency benefits all members of the project to know more about each other’s work, just like the initial thoughts of open-source techs. The CKAN community played a crucial role in helping us navigate challenges, particularly when we encountered errors or needed guidance on best practices. This highlighted the value of open-source ecosystems, where collaborative efforts lead to better software and knowledge sharing.

We also gained a deeper understanding of CKAN’s API, which is a powerful tool for automating dataset management and integrating CKAN with other systems. Working on both the CKAN API library and direct HTTP requests allowed us to compare different methods of interacting with CKAN and reinforced the importance of error handling and security.

Despite of the steep learning curve, the experience was highly rewarding. We not only expanded our technical skill set but also developed a stronger appreciation for open-source and extensible systems like CKAN. Moving forward, we recommend further improvements in CKAN’s user interface, particularly for non-technical administrators, and better integration of extension management tools to simplify the customization process.

In conclusion, this project successfully demonstrated CKAN’s capabilities as a data management platform and provided an example and foundation for further exploration and development.

References

1. https://www.godaddy.com
2. https://www.oracle.com/cloud/
3. https://github.com/DiaoFu/Accelerating/blob/main/Documents/Ckan%20install%20and%20demonstration/ckan_install.pdf
4. https://github.com/DiaoFu/Accelerating/blob/main/Documents/Ckan%20install%20and%20demonstration/ckan_manage_back.pdf
5. https://github.com/DiaoFu/Accelerating/blob/main/Documents/Ckan%20install%20and%20demonstration/ckan_manage_front.pdf
6. https://github.com/DiaoFu/Accelerating/blob/main/API/API_Demo.ipynb
7. https://github.com/DiaoFu/Accelerating/blob/main/Documents/Participating%20in%20the%20CKAN%20community/Participating%20in%20the%20CKAN%20community.pdf