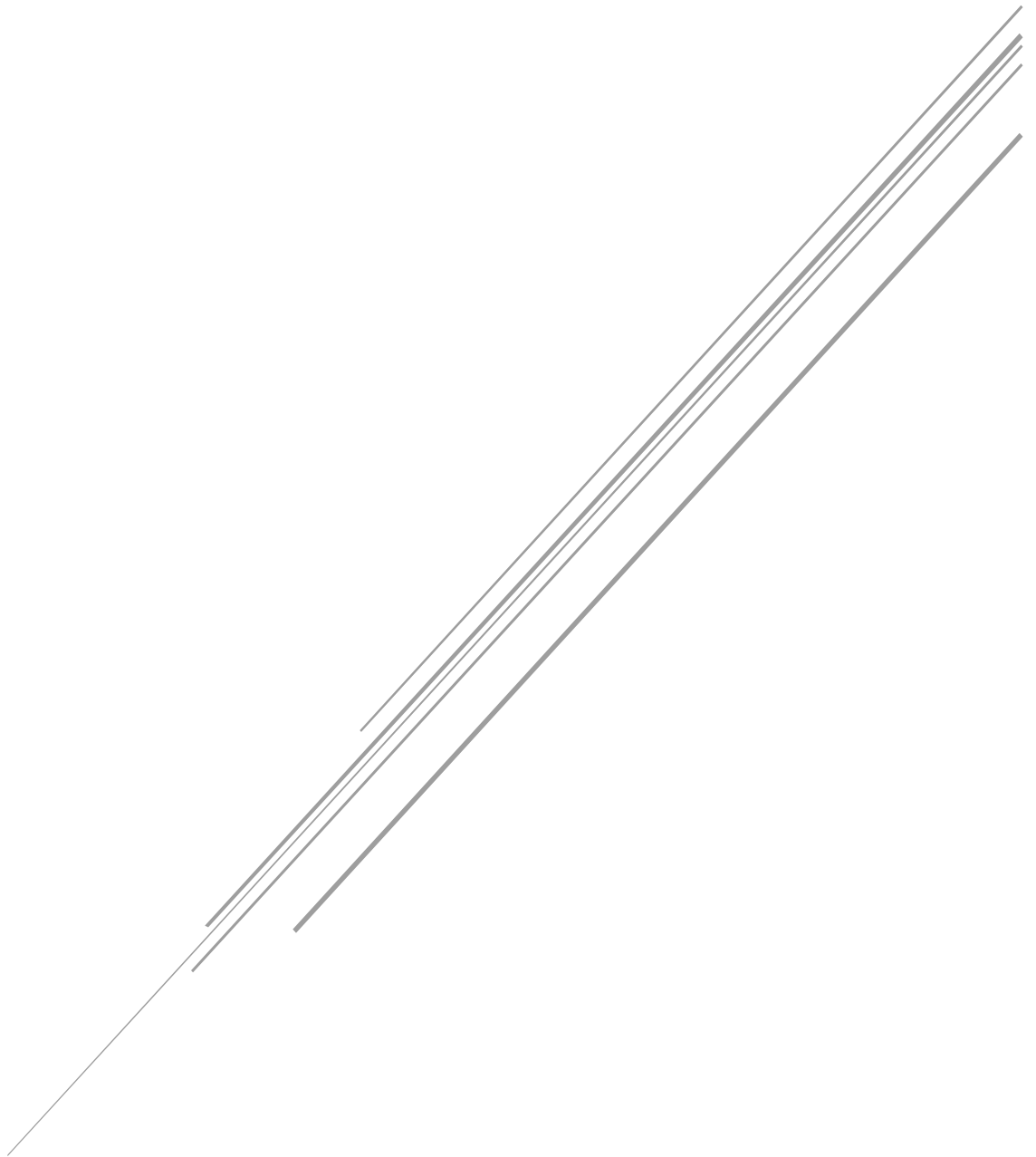


BUILDING SCALABLE POWER APPS

General Guidance v0.4 (Curated Draft)



CONTENT MAP

Evaluate

- We want to build an app for 50K – 500K users with 500 – 5K concurrent users. Is this possible? If yes, which data sources can I use?
- What are the limits (tenant / environment / app / user) we need to keep in mind while evaluating the app for each data source / connector?

Design

- What are the design best practices we should follow for such large-scale apps?
- What are the common performance Issues and how can we address them while we design/develop?

Develop

- What tools are available at development stage to develop for scale and development?

Test

- Once we have built app, how can we load test it to ensure that it scales and performs as expected?

Monitor

- Once in production, how do we proactively monitor performance?

Resolve

- When facing performance issues, how can we troubleshoot and resolve?

INTRODUCTION

The performance of an app is important to keep users happy. Apps can go from mediocre to great just based on performance. And sometimes it can be as simple of a change as caching data in a collection or removing redundant calls to the data source.

The most common app performance problems come from interactions with data sources. Almost every app has one or more data source. Power Apps natively supports over 200 different connections to these data sources and using these connections is key to a great app.

Calling these data sources from your app is often the largest bottleneck in your app because of the time it takes to call across the network to the data source, process the request on the data source side, return the data to Power Apps across the network, and then for Power Apps to process and display the data. Optimizing these interactions with data sources is key to great performance.

EVALUATE

Even before building your app, evaluate if app requirements can be fulfilled by Power App platform for performance and scale.

REQUESTS LIMITS AND ALLOCATIONS

Effective October 2019, to help ensure service levels, availability, and quality, there are entitlement limits to the number of requests users can make each day across Power Apps, Power Automate, AI Builder, Power Virtual Agents, and customer engagement apps (Dynamics 365 Sales, Dynamics 365 Customer Service, Dynamics 365 Field Service, Dynamics 365 Marketing, and Dynamics 365 Project Service Automation).

Requests in Microsoft Power Platform consist of various actions that a user makes across various products. At a high level, below is what constitute an API request:

- **Power Apps** – all API requests to connectors and Microsoft Dataverse.
- **Dataverse (formerly Common Data Service)** – all create, read, update, and delete (CRUD), assign, and share operations including user-driven and internal system requests required to complete CRUD transactions, as well as special operations like share or assign. These can be from any client or application and using any endpoint (SOAP or REST). These include, but are not limited to, plug-ins, classic workflows, and custom controls making the earlier-mentioned operations.

The table below will describe the common requests limits as well as the allocation that a user gets based on the type of license assigned to the user.

Request limits based on user licenses

All the users of Microsoft Power Platform have limits on the number of requests based on the license they are assigned. The following table defines the number of requests a user can make in a 24-hour period:

User licenses	Number of API requests / 24 hours
Dynamics 365 Enterprise applications ¹	20,000
Dynamics 365 Professional ²	10,000
Dynamics 365 Team Member	5,000
Power Apps per user plan ³	5,000
Power Automate per user plan ³	5,000
Office licenses (that include Power Apps, Power Automate, or Power Virtual Agents) ⁴	2,000
Power Apps per app plan	1,000 per app pass

If a user has multiple plans assigned from different product lines, the total number of requests allowed would be the sum of requests allocated to each license type. For example, if a user has both a Dynamics 365 Customer Service Enterprise license as well as a Power Apps per user license then that user will have a total of $20000 + 5000 = 25000$ requests available per 24 hours.

Requests limits not based on licensed users or flows

The Dataverse provides the ability to have identities that do not require any user license to interact with the service. There are four types of these users:

- [Application users](#)
- [Non-interactive users](#)
- [Administrative users](#)
- [SYSTEM user](#)

For these identities, every tenant will get base request capacity per tenant that can only be used by these users and not by users with standard licenses.

This base request capacity is based on the type of subscription, as follows:

- If a tenant has at least one Dynamics 365 enterprise subscription, they will get 100,000 requests per 24 hours.
- If a tenant has at least one Dynamics 365 professional subscription, they will get 50,000 requests per 24 hours.
- If a tenant has at least one Microsoft Power Apps or Power Automate subscription, they will get 25,000 requests per 24 hours.

After base request capacity is exhausted, customers can increase this capacity by purchasing a Power Apps and Power Automate capacity add-on.

Other applicable limits

Apart from the daily API request limit, there are other service protection limits specific to each service. These limits may be lower or higher than the daily per user limits for a 24-hour period. As with the

daily limits, these limits help maintain the quality of service by protecting the service from malicious or noisy behavior that would otherwise disrupt service for all customers.

Review the following resources for information about *current* service protection limits for each service:

- [Dataverse limits](#): applicable for model-driven apps and customer engagement apps (such as Dynamics 365 Sales and Customer Service), Power Apps, and Power Automate connecting to Dataverse/customer engagement apps
- [Power Automate limits](#): applicable for automated, scheduled, and instant flows
- [Limits in connectors](#): applicable for Power Automate and Power Apps

What tools can I use to monitor and analyse API requests across the platform?

Today, the Power Platform admin center contains reports on Dataverse API requests. This reporting today accounts for interactive and non-interactive traffic. This helps you to quickly view adoption and user metrics for your organization. If your apps or flows primarily use the Dataverse, then these reports can serve as good approximations of the total usage of your solutions.

Additionally, for Power Automate usage specifically, you can see the action usage for a given flow by selecting the Analytics action from the flow properties page, and this works across all types of actions. However, if your apps or flows do not use the Dataverse, then there are no reports available in the Power Platform admin center at this time.

What happens if a user or integration exceeds request capacity?

When users exceed their limits, administrators can see this in the admin center (see above). You can do either one of the following:

- Adjust the app or flow to use fewer API requests
- Purchase the **Power Apps and Power Automate capacity add-on** for your organization.

Users won't be blocked from using an app or flow for occasional and reasonable overages at this point in time. However, if a user or flow exceeds the limits consistently for an extended period of time (more than 14 days), that user may be disabled or flow turned off.

DATAVERSE LIMITS

There are two categories of limits that apply for Dataverse: *Entitlement* and *Service protection* limits.

Entitlement limits

These limits represent the number of requests users are entitled to make each day. The allocated limit depends on the type of license assigned to each user.

If any user exceeds their request entitlement the administrator would be notified and would be able to assign Power Apps and Power Automate request capacity to that user. Users will not be blocked from using apps for occasional and reasonable overages at this point of time.

User licenses	Number of API requests / 24 hours
---------------	-----------------------------------

Dynamics 365 Enterprise applications ¹	20,000
Dynamics 365 Professional ²	10,000
Dynamics 365 Team Member	5,000
Power Apps per user plan³	5,000
Power Automate per user plan ³	5,000
Office licenses (that include Power Apps, Power Automate, or Power Virtual Agents) ⁴	2,000
Power Apps per app plan	1,000 per app pass
Non-licensed users	See Requests limits not based on licensed users or flows below

Service protection limits

To ensure consistent availability and performance for everyone we apply some limits to how APIs are used with Dataverse. Service protection API limits help ensure that users running applications cannot interfere with each other based on resource constraints. The limits will not affect normal users of the platform. Only applications that perform a large number of API requests may be affected. The limits provide a level of protection from random and unexpected surges in request volumes that threaten the availability and performance characteristics of the Dataverse platform.

We limit the number of concurrent connections per user account, the number of API requests per connection, and the amount of execution time that can be used for each connection. These are evaluated within a five-minute sliding window. When one of these limits is exceeded, an exception will be thrown by the platform.

Because service protection limits are usually only encountered by applications that perform a high volume of data operations, we recommend that developers building those applications apply patterns to retry operations after a period of time when these exceptions are returned. This will allow the application to respond to exceptions the service sends and reduce the total number of requests and achieve the highest possible throughput.

For information about the specific errors that can be returned and how developers can apply patterns to respond to these errors, see [Service Protection API Limits](#).

[Service protection API limits \(Microsoft Dataverse\) - Power Apps | Microsoft Docs](#)

CONNECTORS – MICROSOFT TEAMS

THROTTLING LIMITS

Name	Calls	Renewal Period
API calls per connection	100	60 seconds
Frequency of trigger polls	1	900 seconds
Non-Get requests per connection	300	3600 seconds

[Microsoft Teams - Connectors | Microsoft Docs](#)

CONNECTORS - SHAREPOINT

Name	Calls	Renewal Period
API calls per connection	600	60 seconds

[Overview of the SharePoint connection - Power Apps | Microsoft Docs](#)

[SharePoint - Connectors | Microsoft Docs](#)

REFERENCE

[Service protection API limits \(Microsoft Dataverse\) - Power Apps | Microsoft Docs](#)

[Requests limits and allocations - Power Platform | Microsoft Docs](#)

DESIGN

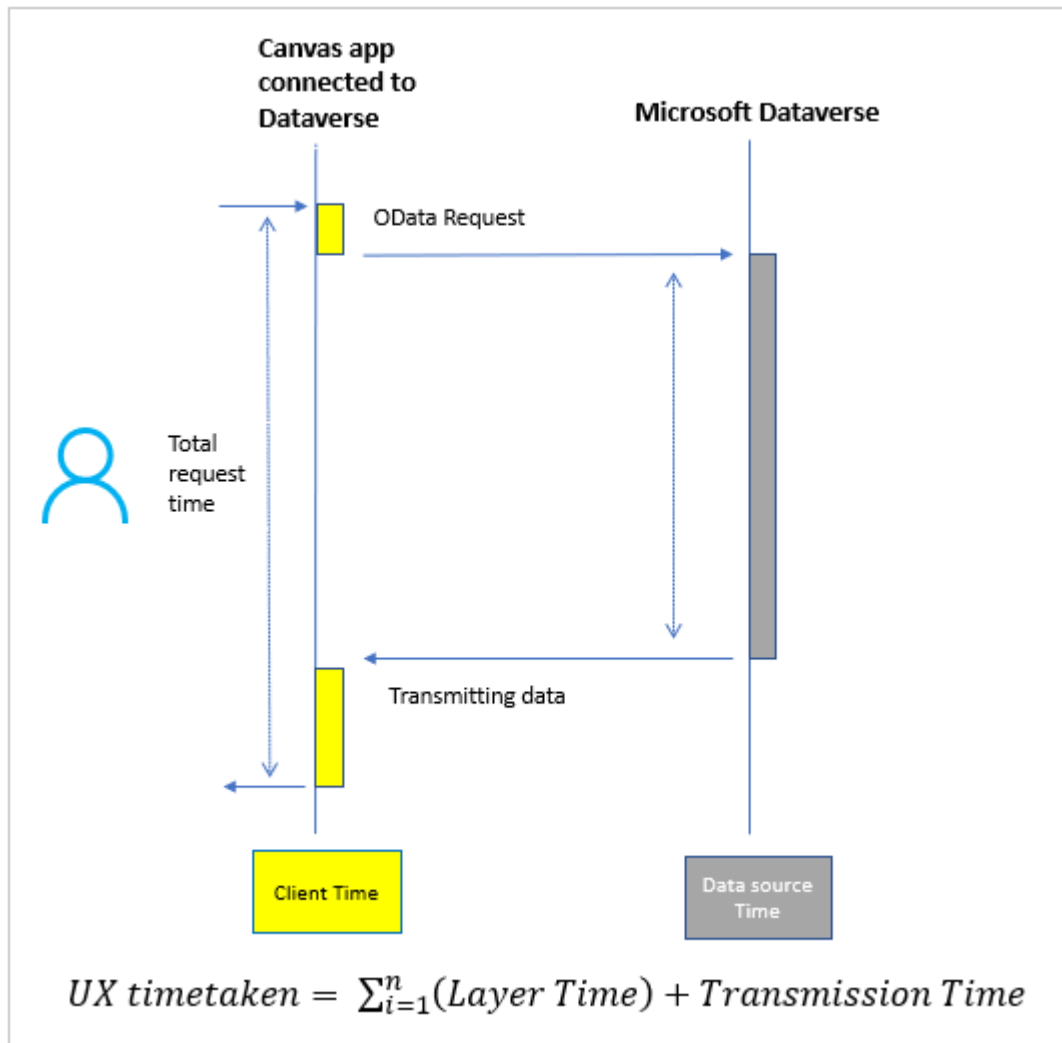
CANVAS APP EXECUTION PHASES AND DATA CALL FLOW

When a user opens a canvas app, the app goes through several phases of execution before showing any user interface. While the app loads, it connects to different [data sources](#)—such as SharePoint, Microsoft Dataverse, SQL Server (on-premises), Azure SQL Database (online), Excel, and Oracle.

A canvas app goes through the following phases of execution before showing the interface to a user:

1. **Authenticate the user:** Prompts the first-time user to sign in with credentials for whatever connections the app needs. If that user opens the app again, that person might be prompted again, depending on the organization's security policies.
2. **Get metadata:** Retrieves metadata, such as the version of the Power Apps platform on which the app runs and the sources from which it must retrieve data.
3. **Initialize the app:** Performs any tasks specified in the [OnStart](#) property.
4. **Render the screens:** Renders the first screen with controls that the app has populated with data. If the user opens other screens, the app renders them by using the same process.

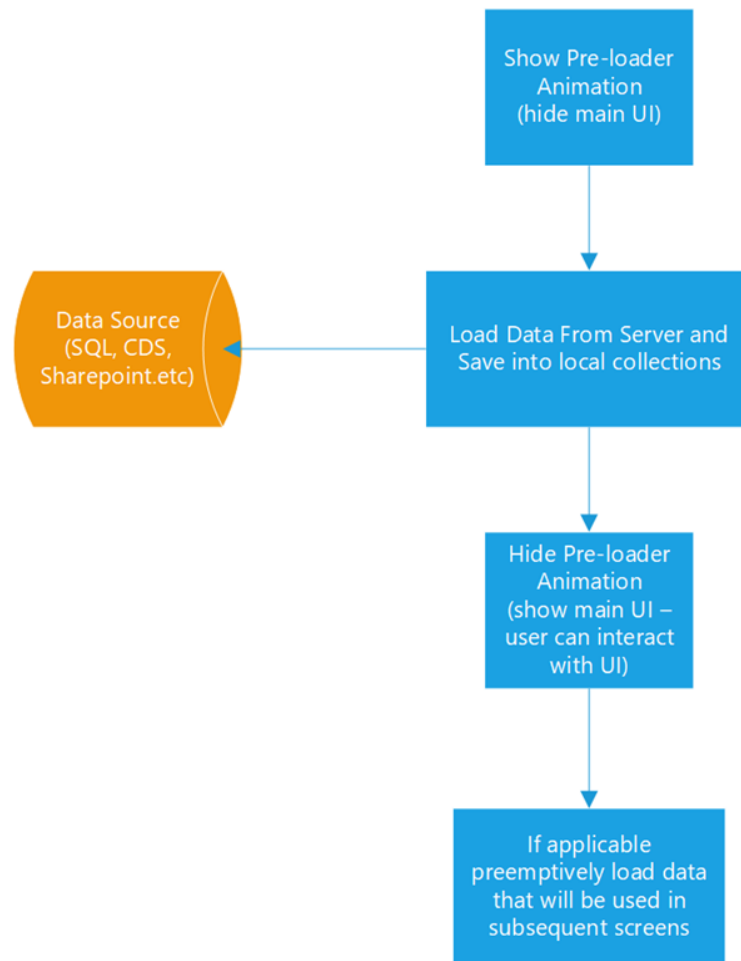
Data call flow with the Common Data Service connector (for Dataverse environments)



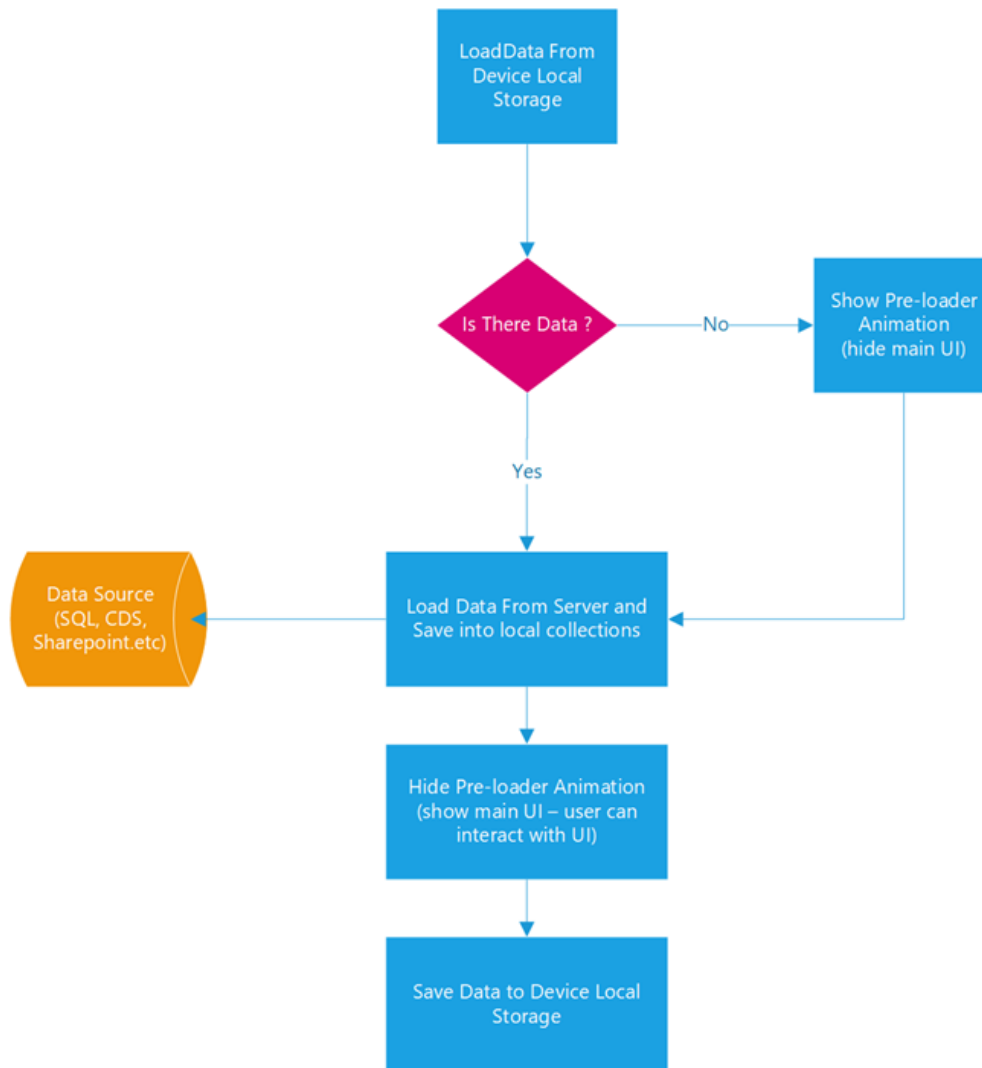
[Canvas app execution phases and data call flow - Power Apps | Microsoft Docs](#)

DATA LOADING PATTERNS

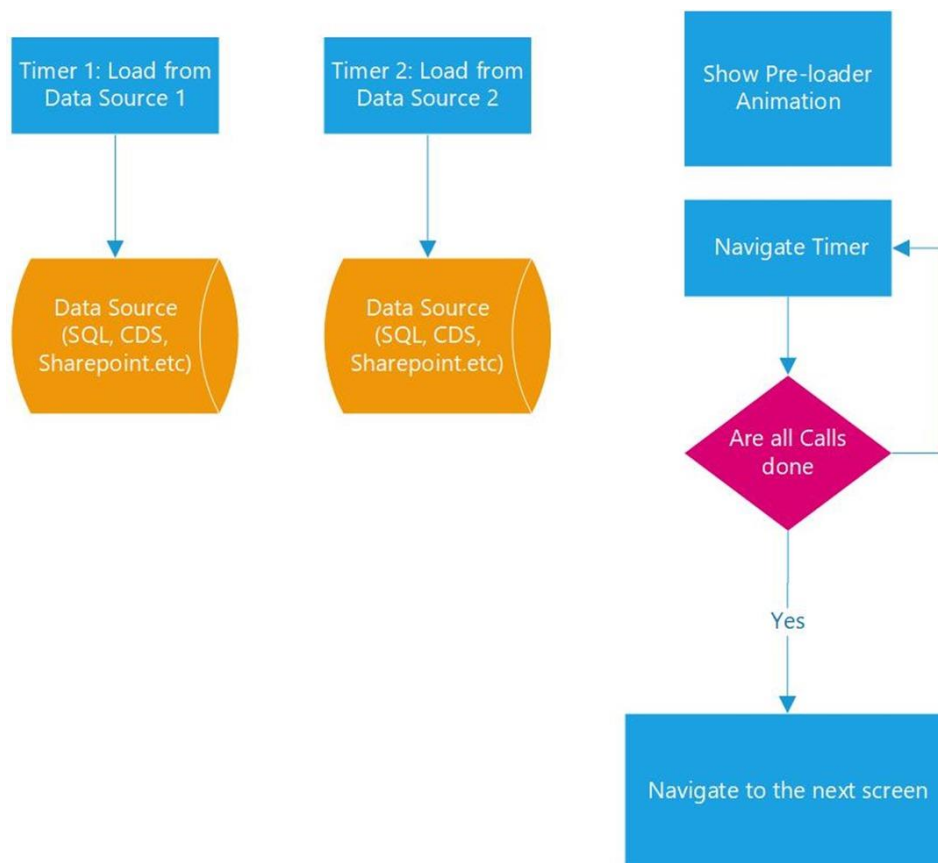
Pattern 1: Load Data from Server



Pattern 2: Load Data From Server with Local caching



Pattern 3: Concurrent Calls



COMMON BOTTLENECKS & RECOMMENDATIONS

TOO MANY REFRESHES

Using the Refresh function, you can force Power Apps to update the data it has gathered from a given data source. This seems like a great function to run because you get the freshest data in your app. But, Power Apps will often handle this refresh for you. For example, when you use a Form to submit a new record to a data source displayed in a Gallery control, Power Apps will automatically refresh that connection. If you include a Refresh function when you navigate to the Gallery screen you are now refreshing the data that Power Apps already refreshed. This is redundant and slows your app down for no reason.

Do not use the Refresh function until you are certain it is needed.

TOO MANY LOOKUPS

As you start to use relational data a common mistake is not to consider the ramifications of a Lookup function inside of a Gallery. When you place a Lookup function on a Label inside of the Gallery, then that Lookup will be performed once for every record in the Gallery. That means if you have 100 records in the Gallery the app has to perform 100 individual Lookup calls to the data source to render. Depending on the data source this could take minutes to render. A better approach is only to display the related data using a details screen or to use a Collection to cache the data from the data source, then the Lookup does not have to execute across the network.

Be careful when making additional calls to remote data sources if you use controls that display multiple records.

STORING DATA IN THE WRONG DATA SOURCE

Different data sources are optimized for different workloads and this should be a consideration when choosing where to store data. One example is storing images or files. A common use of Power Apps is to capture images either using the Camera control or the devices built-in camera app. After the user has taken the image, it needs to be saved. One option is to store the image in the same SQL Server database as the other app data. While possible, it is important to note that SQL Server is very inefficient in storing images. Writing and reading the image file to a SQL database is slow, causing your app to run slow. A better option is to [store Power Apps images in the Azure Blob Storage](#). Azure Blob Storage is much faster than writing the same data to SQL Server. This minor change to the underlying structure of your app can have a useful impact on user satisfaction.

ASSET SIZE

When you are designing your app it is great to include company logos and other visual assets. When you add these assets to your app, make sure the assets are optimized for the size of your app. The higher the resolution of a file, the larger the file size, and the more resources it takes for your app to store and display the image. Use an image editing tool to resize your files to the size you need for your app.

BUILD FOCUSED APPS

Power Apps supports building apps with as many screens as you can imagine, but too many screens is not a good idea. You should build your apps focused on a specific audience and process. This allows you to optimize the user experience for one audience, makes building and troubleshooting the app easier, and reduces the size of the app. If you have one app for everything, consider breaking it into smaller apps by role.

ONSTART

The OnStart event runs when the application is loading and having lots of data called in the OnStart command will slow down the load of the app. If a screen, to be open, has a heavy dependency of controls and values defined on other screens, page load would also be affected by slow screen navigation.

The following are some issues observed in many cases.

1. Many data calls happened within OnStart event which made the app start slow. In enterprise, volume of data calls onto a central data source could drive server bottleneck, resource contention as well.
2. N+1 query problem observed from some galleries and it triggered too many requests to servers.
3. Big latency on OnStart due to heavy scripts. This is a common mistake from many canvas apps. Makers should get only the necessary data from the moment of app start.
4. Too many columns were retrieved.
5. Location of environment vs. end-users is a matter.
6. Whitelist apps.powerapps.com in Firewall. Check Proxy settings of your clients if network proxy configured.

Recommendations

1. Leverage cache mechanism and optimize data calls. Your application would be used by N users at the end. Hence, the number of data calls per user would be landing at the server's endpoints, which could be a spot where bottleneck or throttling could be occurred from.

2. Use View objects in SQL to avoid N+1 query problem or change the UI (user interface) scenarios not to trigger the problem.
3. Use 'StartsWith' instead of 'IN' in formula. If you use SQL data source, for instance, StartWith operator would use index SEEK in SQL database. However, the IN operator would require Index or table scan.
4. Optimize formula in an OnStart event. You can move some formulas to OnVisible event instead. By doing this way, you can let the app start fast and other steps can be continued along with app launching.
5. If you use Microsoft Dataverse, make sure you enabled Explicit Column Selection (ECS) at an advanced setting. Then, Microsoft Dataverse connector will interpret what columns been used in the app and only used columns in the app would be retrieved.
6. We recommend that users should use the new Microsoft Edge instead of IE (Internet Explorer). IE is not fully optimized when it comes to execute JS scripts.
7. Having an environment close to users is also suggested. Although Power Apps has already put in place the Content Delivery Network (CDN) delivering necessary contents of the app from the nearest CDN, data calls would still get the data from the backend data source which might be in different geographical locations. If your app gets a small set of data per request, the impacts would be minimized. Network footprints such as latency, throughput, bandwidth, and packet loss would be another crucial fact affecting performance.
8. Cross check with your network team to make sure *.[PowerApps.com](https://powerapps.com) got whitelisted.

SQL SERVER (ON-PREMISE)

Canvas app can reach out the data out of on-premises SQL via [on-premises data gateway](#). Once on-premises data gateway is configured, Power Apps canvas app can manage data with various on-premises data sources such as SQL, Oracle, SharePoint on on-premises networks.

However, accessing on-premises data sources could suffer from slowness due to the following common causes. Although this topic is focusing on SQL on-premises. They are still valid for other data sources on-premises.

Common causes

1. **Thick client or excessive requests:** some canvas app formed formula to do Group By, Filter By, JOIN operations client-side. Although canvas app can do such operations, they would need CPU and memory resources from client devices. Depending on data size, these operations make extra scripting time at the client side on top of increasing JS heap size of the client. Be aware of each lookup data call also travel to data source via data gateway. In this case, the number of data calls is really a matter.
2. **Unhealthy on-premises data gateway:** As organizations can define multiple nodes of on-premises data gateway, all configured nodes should be healthy, on-premises data gateway service should be up and running. If one of nodes was unreachable, data requests onto the unhealthy node would not return the result within a decent time but 'unreachable' error message after waiting for a while.
3. **Scalability:** In some enterprises, a high volume of data access onto the on-premises data gateway would be expected. In this case, just one node of the on-premises data gateway could be a bottleneck to cover a large volume of requests. A single node of the on-premises data gateway can deal with concurrent connections up to 200. If all these concurrent connections are executing queries actively, other requests would be waiting for an available connection.

Recommendations

1. Do **use the View object in SQL database** for Group By, Filter By, JOIN operations instead of doing such operations at PowerApps client-side. Maker or DBA (Database administrator) can create view(s) with only necessary columns which require for canvas app. Then, use the view entity in canvas app. This approach would also address N+1 query problem.
2. Make sure all on-premises data gateway nodes are healthy and configured at decent network latency between the nodes and SQL instance.
3. In enterprises, having a scalable data gateway cluster would be recommended in case heavy data requests are expected. DBA (Database administrator) can check how many connections get set up between data gateway nodes and the SQL instance. By checking concurrent connections in an on-premises data gateway or in a SQL server, your organization can decide the point when the data gateway should be scaled out how many nodes are.
4. Please do [monitor and optimize on-prem data gateway performance](#) by following instructions in the link. As an on-premises data gateway is in organization's network, Microsoft could not check its performance nor health, but organizations should do.
5. Do profile slow queries in a SQL database and tune if any slow queries are found. That is, tune indexes and queries.
6. Make sure your SQL database has no resource contentions such as CPU bottleneck, IO contention, Memory pressure and/or tempDB contention, apart from checking Locks & Waits, Deadlock and timeout of queries.

Note: Azure SQL provides a feature called **Automatic tuning**. As it is named, it would create missing indexes automatically and fix the execution plan performance problems. Consider turning on this feature on SQL instance.

AZURE SQL

Organizations can connect to Azure SQL Online via SQL connector. In this case, slow requests were caused by slow queries in the database and/or the huge volume of data had to be transmitted to the client. In some case, Service tier of a SQL server was also attributed to slow response.

Common issues

1. **Data size transfer** to client: by default, PowerApps canvas app shows data entities which would be either tables or views from database objects. All columns of entities would be retrieving, which prompts slow response of data requests in case entities have many columns and define many big data types like NVARCHAR(MAX). Simply, total data size of transferring data to client requires transferring time and scripting time to keep that amount data in the JS heap at client side.
2. **Slow queries:** depending on filtering conditions of data requests, the SQL statement which was converted to could be executed with a certain execution plan. If the query executed with heavy IO operation by table scan or index scan, it means data entities might not have proper indexes covering the query. Although the execution plan of queries uses indexes, it could be slow too in case Key Lookup costs high.
3. **Service tier:** Multiple Azure SQL tiers are available. Each tier has a bit different CPU, IO Throughput and IO(Input/output) latency. Under heavy data requests, these resources could be throttled once the threshold hits. Then, query performance would be compromised.

Recommendations

1. **Monitor and turn slow queries.** check this article: [Monitor and Tune for Performance](#). [Query Store](#) would also provide the necessary information to find slow queries. You can use [Extended Events](#) to trace SQL. If you need more details, please refer to [Quick Start: Extended events in SQL Server](#) and SSMS XEvent Profiler.

2. **Do use View** object in Azure SQL online for Group By, Filter By, JOIN operations instead of doing such operations at PowerApps client-side. In addition, View can define only necessary columns. View can select columns and remove some big data type like NVARCHAR(MAX), VARCHAR(MAX) and VARBINARY(MAX) unless necessary. Maker or DBA (Database administrator) can create view(s) with only necessary columns which require for canvas app. Then, use the view entity in canvas app. This approach would also address N+1 query problem.
3. **Check the service tier of Azure SQL online if it is on DTU-Based purchase model.** Lower tier would have some limitations and constraints. From a performance perspective, CPU, IO throughput and latency would be matter. Hence, check the performance of the SQL database and check if resource usage exceeds the threshold or not. on-premises SQL normally sets the threshold of CPU usage on around 75%, for example.

SHAREPOINT ONLINE

SharePoint connector pipelines to SharePoint list(s).

Common issues

1. **Data size** transmitting back to client is matters, especially when the SharePoint data source is remote. If formula in events at canvas app has nondelegable functions inside, Power Apps platform would retrieve records up to Data Row Limits, default 500 but maker can change it up to 2000. If Data Row Limits were set to 2000 and the SharePoint list has many columns, data size transmitting to client could be huge and it could lead to slowness.
2. **Too many dynamic lookup columns:** SharePoint supports various data types including dynamic lookup, Person or Group and Calculated. If a SharePoint list defines too many dynamics columns, it would take time to manipulate these dynamic columns within SharePoint itself before serving asked data requests. This would depend on the volume of data rows on the SharePoint list.
3. **Picture column and Attachment:** size of image and attached file will attribute to slow response if they are all retrieving to client unless specific columns specified.

Recommendations

1. As SharePoint provides many [delegable functions](#), it is worthy checking your formula to see if it would be delegable or not. Otherwise, PowerApps would retrieve the number of records to client, which defined within Data Row Limits (Default 500), and then apply formula on a retrieved data set at client side. **Not only reducing Data Row Limits to a low value or at least staying at the default but also forming the formula to be delegable are important to make the app performant.** For instance, let say you have an ID column defined Number data type in the list. Both formulas below will return the results as expected. However, the former is nondelegable and the latter delegable.

Formula	Delegable?
Filter ('SharePoint list data source', ID = 123)	Yes
Filter('SharePoint list data source', ID ="123")	No

As we assume that the ID column in SharePoint defined data type as Number, right-hand side value should be numeric variable instead of string variable. Otherwise, this type of mismatch would trigger the formula to be nondelegable.

2. Review your SharePoint list and make sure only the necessary columns have been defined. As number of columns in the list would affect performance of data requests because either matched records or records up to data low limits would be retrieving and transmitting back to client with all columns defined in the list whether the app uses some or not. Enabling Explicit Column Selection (ECS) is highly recommended so that data requests would ask only used columns on the app.
3. Do not overuse dynamic Lookup columns and Person or Group type in SharePoint. Otherwise, extra overheads would be seen on the SharePoint side to manipulate data before applying any filter or search on. You can use a static column to keep email aliases or names of people instead.
4. If you have a gigantic list having hundreds of thousands of records, consider partitioning the list to split into several ones per category or datetime. For instance, your data could be stored on different lists on a yearly or monthly base. Then, you can implement the app to let a user select a time window to retrieve data within that range.

Within a controlled environment, the performance benchmark has proved that the performance of OData requests against SharePoint list were highly related to the number of columns in the list and the number of rows retrieving limited by Data Low Limits. The lower column and the lower data row limits setting perform the better. In the real world, however, it is quite hard to simply reduce data rows limits and columns because the app needs a certain amount of data to cover business scenarios. Hence, please monitor OData requests at the client side and tune these two knobs.

MICROSOFT DATAVERSE

Microsoft Dataverse provides a handy way to define custom entities with built-in security model where you can securely store your business data in.

Common issues

1. Too much data transmitted to a client also made requests be slow. For instance, if your app has set Data Row Limits to 2000, instead of default 500, it adds up extra overhead on transferring data and manipulating received data to JS Heap at client side.
2. The app did run client-heavy scripting such as Filter By/Join at client side instead of doing such operation at server side.
3. Canvas app had used old commondataservice connectors. Firstly, the old commondatasource connectors got some overheads. Hence, OData requests via the connector were slower than that via Microsoft Dataverse connector.

Recommendations

1. **Enable Explicit Column Selection (ECS)** which would select only used columns in your app instead of retrieving all columns of the entity you used in your app.
2. **Leverage Microsoft Dataverse View.** Microsoft Dataverse View makes a logical view out of entities with joining/ filtering entities. For instance, if you should join entities and filter their data, you can define a view via Microsoft Dataverse View designer by joining them and define only necessary columns. Then, use this view in your app which put load to server by avoid the app from joining overhead at client side. This would reduce not only extra operations but also data transmission.
3. Reduce Data Row Limits to 500 at least. Please think about your app really requires retrieving more than 500 records or not. As your app might be running at mobile/tablet devices, having light-weight data at clients would perform better. In many cases, delegable functions cover your business logic.

POWER APPS COMPONENT FRAMEWORK

1. If you wish to access a large chunk of data items, implement [filtering](#) and [paging](#) methods to limit how much data should be loaded.
2. Batch your network calls, and make any network calls asynchronous. Design the component in such a way that all the required information is provided with a single call.
3. Design the component such that the user has to perform an action (such as clicking on a button) to initiate the loading of a specific item's data rather than making the call for each data item.
4. Ensure that you clean up the resources using the [destroy](#) function. Open network calls, connections, event handlers, and references to DOM nodes need to be cleaned up to increase the performance.

DEVELOP

SOLUTION CHECKER

The solution checker is a tool that checks whether the solution you've created is healthy. You can quickly review issues and see recommended fixes. With the solution checker feature, you can perform a rich static analysis check on your solutions against a set of best practice rules and quickly identify these problematic patterns. After the check completes, you receive a detailed report that lists the issues identified, the components and code affected, and links to documentation that describes how to resolve each issue.

The solution checker analyzes these solution components:

- Dataverse plug-ins
- Dataverse custom workflow activities
- Dataverse web resources (HTML and JavaScript)
- Dataverse configurations, such as SDK message steps

More information: [Use solution checker to validate your model-driven apps in Power Apps](#)

IMPROVE PERFORMANCE

Some of the common techniques you can apply to mitigate those performance issues.

REPUBLISH YOUR APP

The Power Apps team is constantly updating Power Apps to bring new features and to increase performance. The only way that your app takes advantage of these advancements is for you to open the app and publish again. Your app will stay on the version that it was published on until you do this. So periodically revisiting your app to move to the latest version will provide you with the best possible performance.

USE COLLECTIONS TO CACHE DATA

Often in your app you will find yourself query the same data repeatedly, like in the case of pulling the lists of departments for your drop-down menus. In those instances, you can query for the data once and then reuse that data throughout the app. This reduces the repetitive calls to the data source across the network. The following is an example of this process.

One thing to keep in mind is that the Collect function is not delegable. By default, that means only the first 500 items will be returned from your data source and stored in the collection. Be sure to plan for this limitation as you implement the use of collections in your app.

DELEGATION ALSO AFFECTS PERFORMANCE

When a formula delegates to the data source, all of the processing is handled by the data source, and only the matching rows are returned across the network to the app to be displayed. If a function is not delegable, then it is very common to change the delegation limit to 2,000 rows. This means that the first 2,000 rows will be downloaded across the network and then processed locally. In scenarios where you

are on a slow cellular connection or a low-end mobile device this processing can take a considerable amount of time, causing a poor experience for the user.

Try to use only delegable functions as much as possible. If your function is not delegable, then plan for the impact on the end user.

USE THE CONCURRENT FUNCTION TO LOAD MULTIPLE DATA SOURCES

The **Concurrent** function evaluates multiple formulas at the same time. Normally, multiple formulas are evaluated by chaining them together with the `:` operator, which evaluates each sequentially in order. When the app performs operations concurrently, users wait less for the same result.

In the [OnStart](#) property of your app, use **Concurrent** to improve performance when the app loads data. When data calls don't start until the previous calls finish, the app must wait for the sum of all request times. If data calls start at the same time, the app needs to wait only for the longest request time. Web browsers often improve performance by performing data operations concurrently.

PREVIEW FEATURES

Preview features are features that have been well tested and are close to being released. They will be available for all apps soon. Testing and understanding these features helps you to prepare for when they become standard, and most are enabled by default in new apps.

An example of a current preview feature that helps to increase performance is Delayed load. This feature "Speeds up your app's start time by setting on-demand screen expression calls", which means that screens aren't processed until the user accesses the screen, making the app start and run faster.

EXPERIMENTAL FEATURES

Experimental features are features that might change, break, or disappear at any time. The features are off by default. You can sometimes find performance features here as well, so it is worth taking a look. However, keep in mind you do run a risk by incorporating them into production apps as they can evolve, completely change, or disappear.

An example of a current experimental feature that you might experiment with is "Use longer data cache timeout and background refresh". This feature increases the timeout from 30 seconds to 1 minute for some of the common data sources allowing for more loading time. This might not make for the best user experience but perhaps if you find a way to do this in the background it could let you work with slower queries.

ONSTART VERSUS ONVISIBLE

OnStart and OnVisible are part of your toolkit for building great apps, but from a performance point of view, they can have a major impact.

- OnStart - This is an app-level property. Formulas in this property are run once, when the app starts, and never again. All of the formulas must process before the app opens. This is often used to initialize data that you will need throughout the app.
- OnVisible - This is a per-screen property. Formulas in this property are run every time a user navigates to the screen. The screen will render before the formula is finished processing.

In most apps, you use a mixture of OnStart and OnVisible to get the optimal experience.

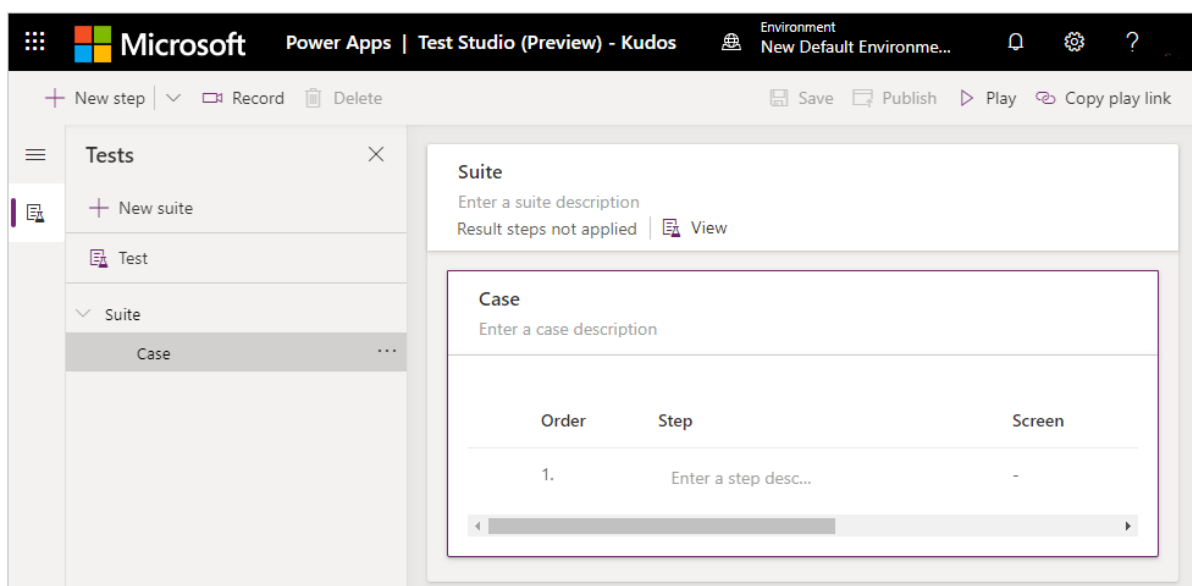
TEST

TEST STUDIO

Build end-to-end UI tests for your canvas app using Test Studio. Maintain your app quality by continually validating that your app works as expected when new changes or updates are deployed.

Power Apps Test Studio is a low-code solution to write, organize, and automate tests for canvas apps. In Test Studio, you can write tests using Power Apps expressions or use a recorder to save app interaction to automatically generate the expressions. You can play written tests back within the Test Studio to validate app functionality, and also run the tests in a web browser and build the automated tests into your app deployment process.

[Test Studio for testing canvas apps - Power Apps | Microsoft Docs](#)



You can run your canvas app tests built in Test Studio using the [Azure Pipelines classic editor](#) in [Azure DevOps Services](#).

You can use a public project on GitHub - [Microsoft/PowerAppsTestAutomation](#) to:

- Automate operations of signing in to your application.
- Open a browser on the build agent and execute a set of test cases and suites.
- View the status of the test execution in the Azure DevOps pipeline.

[Automate tests with Azure Pipelines using classic editor - Power Apps | Microsoft Docs](#)

MONITOR

When you're testing for performance issues, you can use Monitor to check network activity, similar to a network trace in the browser.

[Introducing Monitor to debug apps and improve performance | Microsoft Power Apps](#)

EASYREPRO

EasyRepro is the tool provided for Dynamics 365 and Power Apps model-driven apps. It not only includes a testing tool but also has over 200 sample test cases to help you speed up the testing process. For more information, see the blog post [EasyRepro automated testing framework](#), and access it at the [EasyRepro GitHub repository](#).

USING THE TIMER CONTROL TO GET METRICS

When it comes to working with data connections for retrieving or uploading data, hard numbers are helpful. In Power Apps, you can use a variable and a Timer control to capture how long your formula takes to run. You could log this data to a different collection and then average the numbers to determine how long it takes. You can also apply this concept to submitting data. Remember to test not only from your local computer but from all of the scenarios for your user's environments.

TEST WHERE YOUR USERS WILL USE THE APP

This is more advice than technique. For most app builders the best place to run an app is from the local PC they use to build the app. Testing from that machine generally gives the best case results but may not match your user's experience. Far too often they forget to test the way the user will run the app. For example, if you are going to build a mobile app that runs over a cellular network then make sure your testing includes the same. Understanding the smaller form factor of the mobile device and the latency of a varying internet connection needs to factor into your app's design. The previous timer testing method is great here. Compare the app's query or upload performance between your PC, your mobile phone on Wi-Fi, and your mobile phone on cellular data. Determine if you are satisfied with all three scenarios or if you need to optimize your app for the slower network.

USE LABELS TO HELP WITH TESTING

As your app incorporates more complex logic and more behind the scenes variables to facilitate that logic consider using label controls as part of your testing toolkit. Simply adding a label to the screen that displays the value of the variable can go a long way to helping you understand why your app is or is not doing something. You can use this during the building and testing phase. When your app is live, you can add additional functionality for hiding and showing these troubleshooting tools.

While you are in the Power Apps Studio you can also select **File** and **Variables** to see all of your variables, their values, where they were created, and where they are used in the app.

Another way to use labels during the build process is to add a label to the welcome screen where you manually display a version number. Power Apps caches your app to optimize your experience. When you are publishing repetitively, like when customizing a SharePoint form, it can be confusing to know which version of the form you are seeing as you may see a cached version. By simply adding a label with v1 or v2 in the corner, you will always be able to check the version.

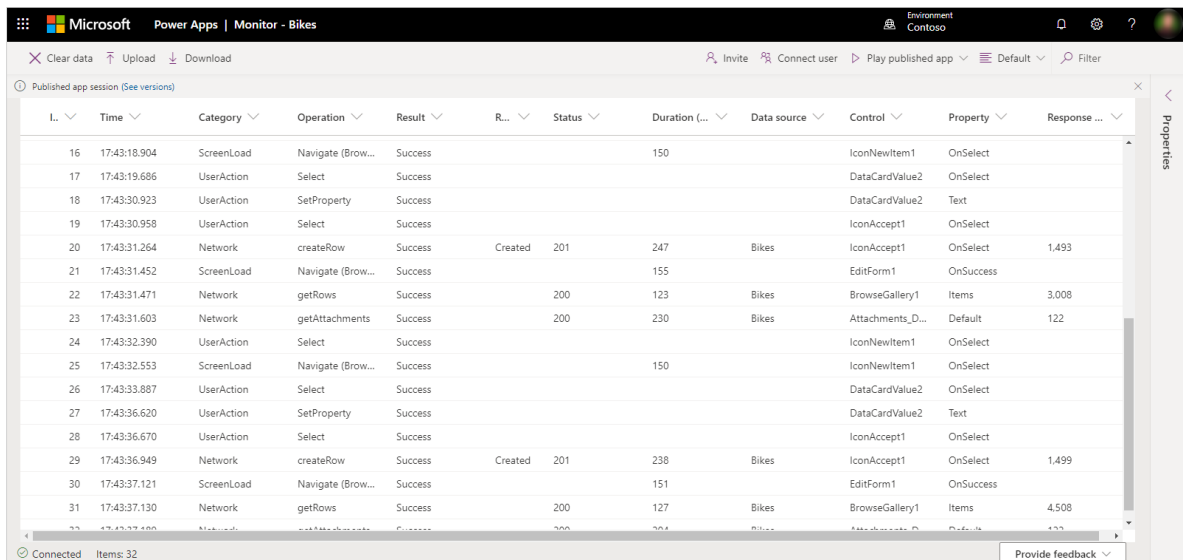
LOOKING AT THE NETWORK ACTIVITY OF YOUR APP

To look at actual network calls and performance, you can use your browsers built-in developer tools, Ctrl+Shift+I in Chrome or F12 in Edge/IE, or a third-party tool like Fiddler. This will allow you to view the individual network calls made by your app and view details, such as time that each call takes. From a performance point of view, this can be valuable.

MONITOR

MONITOR TOOL

Monitor is a tool that offers makers the ability to view a stream of events from a user's session to diagnose and troubleshoot problems. Makers of canvas apps can use Monitor either to view events while building a new app in Power Apps Studio or to monitor published apps during runtime. Makers of model-driven apps can monitor page navigation, command executions, [form-related issues](#), and other major actions to understand app behavior and make improvements.

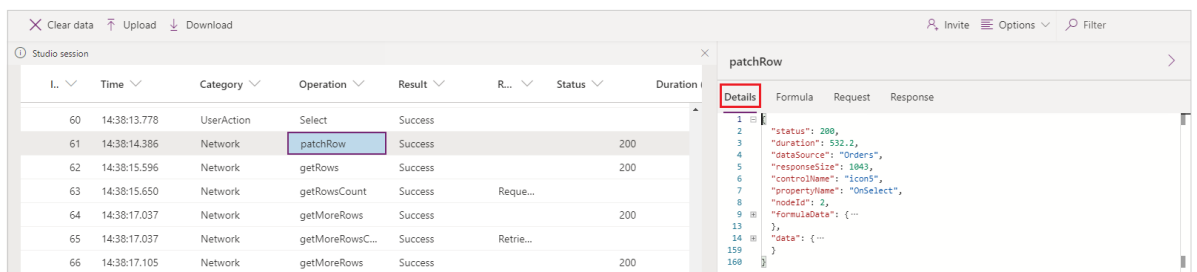


The screenshot shows the Microsoft Power Apps Monitor interface for a published app session. The main area is a table with columns: L., Time, Category, Operation, Result, R..., Status, Duration (...), Data source, Control, Property, and Response The table lists various events such as ScreenLoad, UserAction, Network, and getRows, with details like time, category, operation, result, status, duration, data source, control, property, and response. A 'Properties' panel is visible on the right side of the table.

L.	Time	Category	Operation	Result	R...	Status	Duration (...)	Data source	Control	Property	Response ...
16	17:43:18.904	ScreenLoad	Navigate (Brow...	Success			150		IconNewItem1	OnSelect	
17	17:43:19.686	UserAction	Select	Success					DataCardValue2	OnSelect	
18	17:43:30.923	UserAction	SetProperty	Success					DataCardValue2	Text	
19	17:43:30.958	UserAction	Select	Success					IconAccept1	OnSelect	
20	17:43:31.264	Network	createRow	Success	Created	201	247	Bikes	IconAccept1	OnSelect	1,493
21	17:43:31.452	ScreenLoad	Navigate (Brow...	Success			155		EditForm1	OnSuccess	
22	17:43:31.471	Network	getRows	Success		200	123	Bikes	BrowseGallery1	Items	3,008
23	17:43:31.603	Network	getAttachments	Success		200	230	Bikes	Attachments_D...	Default	122
24	17:43:32.390	UserAction	Select	Success					IconNewItem1	OnSelect	
25	17:43:32.553	ScreenLoad	Navigate (Brow...	Success			150		IconNewItem1	OnSelect	
26	17:43:33.887	UserAction	Select	Success					DataCardValue2	OnSelect	
27	17:43:36.620	UserAction	SetProperty	Success					DataCardValue2	Text	
28	17:43:36.670	UserAction	Select	Success					IconAccept1	OnSelect	
29	17:43:36.949	Network	createRow	Success	Created	201	238	Bikes	IconAccept1	OnSelect	1,499
30	17:43:37.121	ScreenLoad	Navigate (Brow...	Success			151		EditForm1	OnSuccess	
31	17:43:37.130	Network	getRows	Success		200	127	Bikes	BrowseGallery1	Items	4,508

When you select an event in the grid, a panel displays additional details about the event. The panel has four tabs:

- **Details:** Shows a high-level overview of the event that you select. Some of the data might be collapsed in the tree view. You can expand and drill down to view content.



The screenshot shows the Microsoft Power Apps Monitor interface with the 'Details' panel open for a selected event. The panel has four tabs: Details, Formula, Request, and Response. The 'Details' tab is active, showing a tree view of the event details. The 'Formula' tab shows the formula for the event. The 'Request' tab shows the HTTP request that was sent. The 'Response' tab shows the HTTP response that was received.

L.	Time	Category	Operation	Result	R...	Status	Duration (...)
60	14:38:13.778	UserAction	Select	Success			
61	14:38:14.386	Network	patchRow	Success			200
62	14:38:15.596	Network	getRows	Success			200
63	14:38:15.650	Network	getRowsCount	Success	Reque...		
64	14:38:17.037	Network	getMoreRows	Success			200
65	14:38:17.037	Network	getMoreRowsC...	Success	Retrie...		
66	14:38:17.105	Network	getMoreRows	Success			200

- **Formula:** Shows the related formula from your app for the selected event. The name of the control property triggering the event is displayed on top of the tab and inside the event table.
- **Request:** Shows the HTTP request that was sent.
- **Response:** Shows the HTTP response that was received. You can view the response in JSON format.

[Introducing Monitor to debug apps and improve performance | Microsoft Power Apps](#)

SUPPORTED EVENTS

Canvas apps	Model-driven apps
<ul style="list-style-type: none">• Data connectors• Network events (error status codes highlighted)• Screen load metrics• Cross-screen dependency warning• User actions such as <i>Navigate</i>, <i>Select</i>, and <i>SetProperty</i>• Custom Trace()• Delegated versus non-delegated queries• Verbose switch (internal telemetry)• Delegation• Function• Network• Parsing• Performance• Scenario• ScreenLoad• Telemetry• UserAction• Verbose	<ul style="list-style-type: none">• Form events, such as load and save• Network• Page navigation• Command executions

[Overview of Power Apps with Monitor - Power Apps | Microsoft Docs](#)

OFFLINE ANALYSIS

You can **download** the events listed in [Supported events](#) for offline analysis. Events can be downloaded in .json or .csv format, and you can share them with others. The .csv files can only be downloaded, not uploaded, but if you download the events in .json format, you can upload them later into Monitor for analysis. You can also attach a trace file to support service requests, which can help speed up getting the solution to your problem.

<https://docs.microsoft.com/en-us/powerapps/maker/monitor-advanced#download-and-upload-trace-files>

Collect / Analyze logs using Application Insights & Power BI

You can connect your app with [Application Insights](#), a feature of [Azure Monitor](#). Application Insights includes powerful analytics tools to help you diagnose issues and to understand what users actually do with your app.

You can export your Application Insights data and query results to Power BI for analysis and data presentation.

[Analyze app telemetry using Application Insights - Power Apps | Microsoft Docs](#)

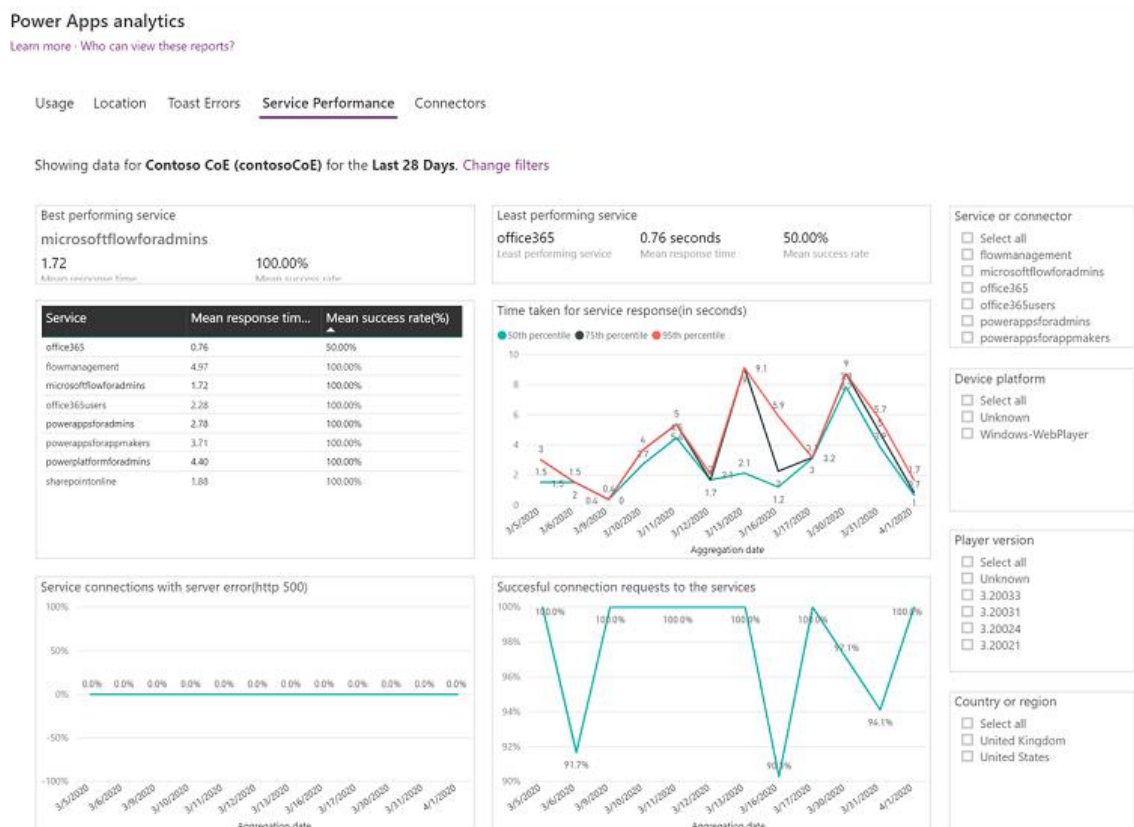
ADMIN ANALYTICS FOR POWER APPS

Analytics for the environment admin is available at the Microsoft Power Platform admin center. The admin reports provide a view into environment level usage, errors, service performance to drive governance, and change management services to users. These reports are available for canvas apps only and not available for model-driven apps.

Service Performance report provides details of all standard and custom connectors to understand performance bottlenecks and client versus service API issues. An environment admin will get insights into:

- Connectors used in the environment.
- Best and least performant service and the API service response times.
- Success rates for each service to determine areas that need attention.
- The 50th, 75th, and 90th percentile response times for each service.
- The number of HTTP 500 error codes of connectors indicating issues around the server not responding to calls from the client.
- The number of successful connection requests.

All the service performance KPI's can be filtered with attributes like a specific service or connector, device platform, player version, and country, state, or city to drill down into the specific API.



[Administrator analytics and reports for Microsoft Power Apps - Power Platform | Microsoft Docs](#)

RESOLVE

STARTUP ISSUES

[Troubleshooting startup issues for Power Apps - Power Apps | Microsoft Docs](#)

FURTHER READING

[The importance of thinking about performance - Learn | Microsoft Docs](#)

[Considerations for optimized performance in Power Apps | Microsoft Power Apps](#)

[Performance considerations with PowerApps | Microsoft Power Apps](#)

[PowerApps canvas app coding standards and guidelines.pdf \(windows.net\)](#)