Dashboard Migration Guide

Table of Contents

- Need Help?

Updating to Use Your utils.py

Overview

Your `utils.py` file contains all the necessary functions. Here's how to update each dashboard file to use it.

Key Changes Required

1. Import Statements

OLD (My modules):

```
from modules.aws_helper import AWSConfig, AWSOrganizations, AWSSession
from modules.sidebar_simple import render_sidebar
```

NEW (Your utils.py):

```
from utils import (
    assume_role,
    setup_account_filter,
    get_account_name_by_id,
)
import boto3  # Add this for creating clients
```

2. Sidebar Function

OLD:

```
account_ids, regions = render_sidebar(page_key="securityhub")
```

NEW:

```
account_ids, regions = setup_account_filter(page_key="securityhub")
```

3. Getting AWS Clients

OLD (My helper):

```
client = AWSSession.get_client_for_account('securityhub', account_id, role_name, region)
```

NEW (Your pattern):

```
def get_securityhub_client(account_id, role_name, region):
    """Get Security Hub client using your assume_role function"""
    credentials = assume_role(account_id, role_name)
    if not credentials:
        return None

    return boto3.client(
        'securityhub',
        region_name=region,
        aws_access_key_id=credentials['AccessKeyId'],
        aws_secret_access_key=credentials['SecretAccessKey'],
        aws_session_token=credentials['SessionToken']
    )

# Then use it:<a></a>
sh_client = get_securityhub_client(account_id, role_name, region)
if not sh_client:
    errors.append(f"❌ {account_name}/{region}: Failed to get client")
    return findings, errors
```

4. Getting Account Names

OLD:

```
account_name = AWSOrganizations.get_account_name_by_id(account_id, all_accounts)
```

NEW:

```
account_name = get_account_name_by_id(account_id, all_accounts)
```

5. Role Name

OLD:

```
role_name = AWSConfig.READONLY_ROLE_NAME
```

NEW:

```
role_name = "readonly-role"  # Your role name
```

6. Button Click Pattern

No changes needed! Your `setup_account_filter()` already handles the button click and sets `st.session_state[f"{page_key}_fetch_clicked"] = True`, which is what the dashboards check for.

Complete Updated Files

File 1: Security Hub Dashboard

Changes Made:

1. Import from `utils` instead of `modules`
2. Created `get_security_hub_client()` function using your `assume_role()`
3. Changed sidebar call to `setup_account_filter()`
4. Changed `get_account_name_by_id()` calls
5. Changed role name to `"readonly-role"`

File: `Security_Hub_updated.py` [^149]

File 2: IAM Users Dashboard

Key Changes:

```
# Import<a></a>
from utils import assume_role, setup_account_filter, get_account_name_by_id
import boto3

# Client creation helper<a></a>
def get_iam_client(account_id, role_name):
    """Get IAM client using your assume_role function"""
    credentials = assume_role(account_id, role_name)
    if not credentials:
        return None

    return boto3.client(
        'iam',
        region_name='us-east-1',  # IAM is global
        aws_access_key_id=credentials['AccessKeyId'],
        aws_secret_access_key=credentials['SecretAccessKey'],
        aws_session_token=credentials['SessionToken']
    )

# In helper function<a></a>
def get_iam_users(account_id, account_name, role_name, warning_days, critical_days):
    users_data = []
```

```
    errors = []

    try:
        iam_client = get_iam_client(account_id, role_name)
        if not iam_client:
            errors.append(f"✖ {account_name}: Failed to get IAM client")
            return users_data, errors

        # Rest of your IAM logic...
    except Exception as e:
        errors.append(f"✖ {account_name}: Unexpected error - {str(e)}")

    return users_data, errors

# Sidebar<a></a>
account_ids, regions = setup_account_filter(page_key="iam")
```

File 3: AWS Config Dashboard

Key Changes:

```
# Import<a></a>
from utils import assume_role, setup_account_filter, get_account_name_by_id
import boto3

# Client creation helper<a></a>
def get_config_client(account_id, role_name, region):
    """Get Config client using your assume_role function"""
    credentials = assume_role(account_id, role_name)
    if not credentials:
        return None

    return boto3.client(
        'config',
        region_name=region,
        aws_access_key_id=credentials['AccessKeyId'],
        aws_secret_access_key=credentials['SecretAccessKey'],
        aws_session_token=credentials['SessionToken']
    )

# In helper function<a></a>
def get_config_rules(region, account_id, account_name, role_name):
    rules_data = []
    errors = []

    try:
        config_client = get_config_client(account_id, role_name, region)
        if not config_client:
            errors.append(f"✖ {account_name}/{region}: Failed to get Config client")
            return rules_data, errors

        # Rest of your Config logic...
    except Exception as e:
        errors.append(f"✖ {account_name}/{region}: Unexpected error - {str(e)}")
```

```
    return rules_data, errors

# Sidebar<a></a>
account_ids, regions = setup_account_filter(page_key="config")
```

File 4: Account Health Dashboard

Key Changes:

```
# Import<a></a>
from utils import assume_role, setup_account_filter, get_account_name_by_id
import boto3

# Multiple client creation helpers<a></a>
def get_iam_client(account_id, role_name):
    credentials = assume_role(account_id, role_name)
    if not credentials:
        return None
    return boto3.client('iam', region_name='us-east-1',
                        aws_access_key_id=credentials['AccessKeyId'],
                        aws_secret_access_key=credentials['SecretAccessKey'],
                        aws_session_token=credentials['SessionToken'])

def get_securityhub_client(account_id, role_name, region):
    credentials = assume_role(account_id, role_name)
    if not credentials:
        return None
    return boto3.client('securityhub', region_name=region,
                        aws_access_key_id=credentials['AccessKeyId'],
                        aws_secret_access_key=credentials['SecretAccessKey'],
                        aws_session_token=credentials['SessionToken'])

def get_config_client(account_id, role_name, region):
    credentials = assume_role(account_id, role_name)
    if not credentials:
        return None
    return boto3.client('config', region_name=region,
                        aws_access_key_id=credentials['AccessKeyId'],
                        aws_secret_access_key=credentials['SecretAccessKey'],
                        aws_session_token=credentials['SessionToken'])

def get_backup_client(account_id, role_name, region):
    credentials = assume_role(account_id, role_name)
    if not credentials:
        return None
    return boto3.client('backup', region_name=region,
                        aws_access_key_id=credentials['AccessKeyId'],
                        aws_secret_access_key=credentials['SecretAccessKey'],
                        aws_session_token=credentials['SessionToken'])

# In helper function<a></a>
def get_account_health(account_id, account_name, role_name, primary_region):
    health = {
```

```
        'Account ID': account_id,
        'Account Name': account_name,
    }
    errors = []
    score = 100

    try:
        # IAM check
        iam_client = get_iam_client(account_id, role_name)
        if iam_client:
            # Your IAM logic...
        else:
            errors.append(f"⚠ {account_name}: Cannot access IAM")
            score -= 10

        # Security Hub check
        sh_client = get_securityhub_client(account_id, role_name, primary_region)
        if sh_client:
            # Your Security Hub logic...
        else:
            errors.append(f"⚠ {account_name}: Cannot access Security Hub")
            score -= 10

        # Config check
        config_client = get_config_client(account_id, role_name, primary_region)
        if config_client:
            # Your Config logic...
        else:
            errors.append(f"⚠ {account_name}: Cannot access Config")
            score -= 10

        # Backup check
        backup_client = get_backup_client(account_id, role_name, primary_region)
        if backup_client:
            # Your Backup logic...
        else:
            errors.append(f"⚠ {account_name}: Cannot access Backup")
            score -= 5

        # Calculate final score
        health['Health Score'] = max(0, round(score, 1))

        if score &gt;= 80:
            health['Status'] = '✅ Excellent'
        elif score &gt;= 60:
            health['Status'] = '✅ Good'
        elif score &gt;= 40:
            health['Status'] = '⚠ Fair'
        else:
            health['Status'] = '❌ Poor'

    except Exception as e:
        errors.append(f"✖ {account_name}: Unexpected error - {str(e)}")
        health['Health Score'] = 0
        health['Status'] = '✖ Error'
```

```
    return health, errors

# Sidebar<a></a>
account_ids, regions = setup_account_filter(page_key="health")
```

Template Pattern for New Dashboards

Here's the universal pattern for creating AWS service clients with your `utils.py`:

```
from utils import assume_role, setup_account_filter, get_account_name_by_id
import boto3

def get_[service]_client(account_id, role_name, region):
    """
    Get AWS service client using your assume_role function

    Args:
        account_id: Target account ID
        role_name: Role to assume (typically "readonly-role")
        region: AWS region

    Returns:
        boto3 client or None if assume_role fails
    """
    credentials = assume_role(account_id, role_name)
    if not credentials:
        return None

    return boto3.client(
        '[service_name]',  # e.g., 'ec2', 'rds', 'lambda'
        region_name=region,
        aws_access_key_id=credentials['AccessKeyId'],
        aws_secret_access_key=credentials['SecretAccessKey'],
        aws_session_token=credentials['SessionToken']
    )

def get_service_data_in_region(region, account_id, account_name, role_name):
    """Fetch service data"""
    data_items = []
    errors = []

    try:
        client = get_[service]_client(account_id, role_name, region)
        if not client:
            errors.append(f"✗ {account_name}/{region}: Failed to get client")
            return data_items, errors

        # Fetch your data using the client
        paginator = client.get_paginator('describe_[resources]')
        for page in paginator.paginate():
            for item in page['[Items]']:
                data_items.append({
                    'Account ID': account_id,
                    'Account Name': account_name,
```

```
                    'Region': region,
                    # Your fields...
                })

    except ClientError as e:
        errors.append(f"⚠ {account_name}/{region}: {str(e)}")
    except Exception as e:
        errors.append(f"✖ {account_name}/{region}: Unexpected error - {str(e)}")

    return data_items, errors

# Sidebar<a></a>
account_ids, regions = setup_account_filter(page_key="[your_service]")

# Fetch data<a></a>
if st.session_state.get('[your_service]_fetch_clicked', False):
    # Your fetch logic using the pattern above
    ...
```

Quick Migration Checklist

For each dashboard file:

- [ ] **Change imports**:
  - Remove: `from modules.aws_helper import ...`
  - Remove: `from modules.sidebar_simple import ...`
  - Add: `from utils import assume_role, setup_account_filter, get_account_name_by_id`
  - Add: `import boto3`
- [ ] **Update sidebar call**:
  - Change: `render_sidebar(page_key=...)`
  - To: `setup_account_filter(page_key=...)`
- [ ] **Create client helper functions**:
  - Add `get_[service]_client()` function for each AWS service
  - Use your `assume_role()` inside the helper
  - Return `boto3.client()` with credentials
- [ ] **Update data fetching functions**:
  - Call your new client helper: `client = get_[service]_client(...)`
  - Check if client is None and handle error
  - Use client for API calls
- [ ] **Update account name lookups**:
  - Change: `AWSOrganizations.get_account_name_by_id(...)`
  - To: `get_account_name_by_id(...)`

- [ ] **Update role name**:
  - Change: `AWSConfig.READONLY_ROLE_NAME`
  - To: `"readonly-role"`
- [ ] **Test the dashboard**:
  - Run the dashboard
  - Select accounts and regions
  - Click "Fetch Data"
  - Verify data displays correctly

Example: Complete Migration of EC2 Dashboard

Before:

```
from modules.aws_helper import AWSConfig, AWSOrganizations, AWSSession
from modules.sidebar_simple import render_sidebar

# Sidebar<a></a>
account_ids, regions = render_sidebar(page_key="ec2")

# Helper function<a></a>
def get_ec2_instances(region, account_id, account_name, role_name):
    instances = []
    errors = []

    try:
        ec2_client = AWSSession.get_client_for_account('ec2', account_id, role_name, regi
        # Fetch logic...
    except:
        pass

    return instances, errors

# Fetch<a></a>
account_name = AWSOrganizations.get_account_name_by_id(account_id, all_accounts)
data = get_ec2_instances(region, account_id, account_name, AWSConfig.READONLY_ROLE_NAME)
```

After:

```
from utils import assume_role, setup_account_filter, get_account_name_by_id
import boto3

# Sidebar<a></a>
account_ids, regions = setup_account_filter(page_key="ec2")

# Client helper<a></a>
def get_ec2_client(account_id, role_name, region):
    credentials = assume_role(account_id, role_name)
    if not credentials:
```

```
        return None

    return boto3.client(
        'ec2',
        region_name=region,
        aws_access_key_id=credentials['AccessKeyId'],
        aws_secret_access_key=credentials['SecretAccessKey'],
        aws_session_token=credentials['SessionToken']
    )

# Helper function<a></a>
def get_ec2_instances(region, account_id, account_name, role_name):
    instances = []
    errors = []

    try:
        ec2_client = get_ec2_client(account_id, role_name, region)
        if not ec2_client:
            errors.append(f"❌ {account_name}/{region}: Failed to get client")
            return instances, errors
        # Fetch logic...
    except:
        pass

    return instances, errors

# Fetch<a></a>
account_name = get_account_name_by_id(account_id, all_accounts)
data = get_ec2_instances(region, account_id, account_name, "readonly-role")
```

Summary

Three main changes:

1. **Replace imports**: Use `utils` functions instead of `modules` classes
2. **Create client helpers**: Wrap your `assume_role()` + `boto3.client()`
3. **Update function calls**: Use your function names

Everything else stays the same:

- Session state management
- Parallel processing logic
- Progress bars and status tracking
- Data display and filtering
- CSV export

Your `utils.py` provides all necessary functionality - just need to adapt the calling pattern!

Need Help?

If you encounter issues:

1. **Check credentials**: Make sure `assume_role()` returns valid credentials
2. **Verify role name**: Confirm "readonly-role" exists in all accounts
3. **Test client creation**: Try creating a client manually first
4. **Check error messages**: Enable debug mode to see detailed errors

All dashboard logic remains identical - only the AWS client creation pattern changes!

[1]

⁂

1. utils.py